E-Auction System
Design Document

Document Change Control

Version	Date	Authors	Summary of Changes
2.1	2024/03/10	Kawshar Patel	Added Implementation Changes and Justifications
2.0	2024/03/10	Jacob	Updated group meeting logs
1.3	2024/03/10	Yunfei Cao	Added Activity and Sequence Diagrams
1.0	2024/02/01	Yash Dani	Added Test Cases

Document Sign-Off

Name (Position)	Signature	Date
Kawshar Patel	Kawshar Patel	2024/03/10
Jacob	Jacob	2024/03/10
Yash Dani	Yash Dani	2024/03/10
Yunfei Cao	Yunfei Cao	2024/03/10

Contents

INTRODUCTION

Purpose

This document details the requirements of the system e-Auction. The goal is to develop a full-stack web application using React for the frontend, which communicates with the backend through HTTP/HTTPS responses. The architecture includes a middle layer using Spring Boot, and the backend is implemented in Java with various modules providing services. Following Model-View-Controller(MVC) principles and a microservices architecture, each service can be deployed individually using Docker. The backend connects to the database using a data access layer, utilizing SQL. The system enables users to interact with the application, facilitating an auction environment. The development process will incorporate Git version control, with GitHub serving as the repository for code management. The overall aim of this project is to create a straightforward and user-friendly web application.

Overview

We have navigated through major design choices in this project to build an e-Auction web application. Using React for the front end and Java with Spring boot for the backend, ensuring smooth operations with databases like SQLite and Neo4j. Our design focuses on a client-server architecture, implementing a pub-sub architectural model for the front end and microservices with MVC principles for a strong backend. To visualize our understanding of the system, we have included sequence and activity diagrams for each use case. The architecture section presents a block-arrow figure illustrating the overall system, detailed tables on modules, and the exposed interfaces. We have also included plans for activities, product backlog, and sprint backlog which is supported by group meeting logs and a Gantt chart for clear scheduling. Additionally, we have also focused on the quality part of our systems. We have 17 test cases in the Test Driven Development section to ensure the functionality and reliability of our e-Auction platform. This comprehensive overview lays the foundation of our full-stack web application's structured development and success.

Resources - References

Aroshelova, A. (2022, January 3). Crafting a Solutions Architecture: Foundations of a Real-Time Application Design. Medium. https://medium.com/@aroshelova.tech/crafting-a-solutions-architecture-foundations-of-a-real-time-application-design-dfa3cf7fd16b

1. Educative. (2022, May 31). Software Architecture: Diagramming and Patterns. Educative Blog. https://www.educative.io/blog/software-architecture-diagramming-and-patterns
2. Stack Overflow. (n.d.). Arrow Direction for Fetching Data from Database in Architecture Diagram. Stack Overflow. https://stackoverflow.com/questions/66695185/arrow-direction-for-fetching-data-from-database-in-architecture-diagram

3. Unadkat Jash (2023, June 14). What is Test-Driven Development? BrowserStack Guide. https://www.browserstack.com/guide/what-is-test-driven-development
4. Acunetix. (n.d.). SQL Injection. Acunetix Website Security. https://www.acunetix.com/websitesecurity/sql-injection/#:~:text=Attackers%20can%20use%20SQL%20Injection,delete%20records%20in%20the%20database
5. Gillis Alexander (n.d.). Dependency Injection. TechTarget Search App Architecture. https://www.techtarget.com/searchapparchitecture/definition/dependency-injection#:~:text=Dependency%20 injections%20are%20useful%20to,or%20variables%20in%20an%20object
6. Harvey, J. (2018, August 14). A Beginner's Guide to Creating a Containerized Web Application Development Environment with Docker. Medium. https://medium.com/@jharvey1012/a-beginners-guide-to-creating-a-containerized-web-application-development-environment-with-docker-e903169004ff
7. Michael (2023, May 12). Using Docker in Web Hosting: Complete How-to Guide. SwissMade Host Blog. https://swissmade.host/en/blog/using-docker-in-web-hosting-complete-how-to-guide
8. Lin, Joyce (2019, March 4). Deploying a Scalable Web Application with Docker and Kubernetes. Medium. https://medium.com/better-practices/deploying-a-scalable-web-application-with-docker-and-kubernetes-a5000a06c4e9
9. Kolade, Chris (2023, February 16). SOLID Design Principles in Software Development. freeCodeCamp. https://www.freecodecamp.org/news/solid-design-principles-in-software-development/
10. Creately. (2022, September 2022). UML Diagram Types & Examples. Creately Blog. https://creately.com/blog/diagrams/uml-diagram-types-examples/
11. Girdler, Adriana (2022, November 30). Unified Modeling Language (UML) Tutorial - Use Case, Activity, Class and Sequence Diagrams. YouTube. https://www.youtube.com/watch?v=Lx8INTdIJpY
12. Garcia-Molina, H., Ullman, J. D., & Widom, J. (2009). Database systems: A complete book. Pearson Prentice Hall.

MAJOR DESIGN DECISIONS

Our system will be utilizing React as the frontend framework with Java and springBoot serving as the backend. Our choice of databases includes MySQL. The front end will communicate with the backend through HTTP/HTTPS requests and responses. In order to manage creating, updating, retrieving, and deleting data(CURD) operations in the database, we will have a data access layer between the services and the database using Java and JDBC. This will ensure that we have low coupling between the database and the backend modules which will ensure that changes in the database will not significantly impact the services.

We are planning to construct our system using a client-server architecture. In the front end, which acts as the client, we intend to implement a pub-sub architecture within different react components to maintain loose coupling. This design will ensure that whenever an event occurs it could prompt the rendering of different react components without having them being closely coupled together and reducing dependencies between them. In the backend, which serves as the server, we intend to implement the microservices architecture while incorporating the Model-View-Controller(MVC) layered architecture style. This approach would help us have each module designed in accordance with the single responsibility principle, ensuring that each module has a clear and precise purpose. Within each service, we will have controllers to manage communication, service classes responsible for implementing the main business logic and updating or receiving information from the database, and models for storing data. Each service would also emphasize having low coupled and highly cohesive classes using dependency injection(either through method or constructor injection)(Gillis & Ferguson, 2023). All of the services will communicate with each other over HTTP/HTTPs protocol. Moreover, we will host each service independently using Docker containers which would allow our systems to be scalable.

We are also going to extend our client-server architecture to include a database, with the database acting as a server and the backend as a client. Our coding practices will follow a few principles: each class responsible for one thing (single responsibility), they will be open for extension but closed for modification(open closed), and will depend on abstractions(dependency inversion) so that our classes have single responsibilities, they are modular and have high cohesion low coupling between them(Chris, *Solid design principles in software development* 2023).

For the non-functional requirements, we are prioritizing security and performance. For security, we plan on implementing measures to prevent SQL injection by using parameterized queries which would prevent the attacker from breaking into the databases. Furthermore, we plan to boost performance by optimizing queries through data access techniques like indexing(Garcia-Molina et al., 2009).

In regards to the alternative, we wanted to use the pipe and filter architecture for continuous integration and continuous deployment using CI/CD pipelines. However, our team is not very familiar with DevOps tools and practices, and adopting this architecture will need a significant amount of time and effort. Another option we had was to use the repository architecture for the backend and database, but due to its design, which highly relies on a centralized repository in order to access data, would make the components tightly coupled.

IMPLEMENTATION CHOICES AND JUSTIFICATION

The chosen design patterns and implementation choices contribute to a robust and scalable system architecture for our E-Auction system.

Firstly, the adoption of microservices architecture is crucial for achieving scalability, maintainability, and effective fault tolerance. Each microservice focuses on a specific business need, fostering a modular and single-responsibility design. Additionally, the microservices retrieve data from the database, establishing a client-server architecture where the microservices act as clients, and the database serves as the server. This integration aligns with the principles of microservices architecture, allowing independent development, deployment, and scaling of each microservice while ensuring efficient data retrieval through the client-server model.

To ensure consistency and code reuse across the microservices, a Common Library for shared classes like items, users, and shipping addresses is introduced. Implementing these classes as singletons ensures a consistent representation across services, minimizing resource utilization and preservation of data consistency across services.

The integration between AuctionService and SellerService is designed to promote decoupling and interdependence between these services. In this integration, the SellerService plays an active role by triggering events such as addItem, which adds a new item to the system, and the AuctionService, acting as an observer, reacts to these events by initiating auctions. Rather than having a direct, synchronous interaction between Seller and AuctionService, this implementation takes a more loosely coupled approach and reflects the Observer Pattern which makes the services more modular and allows them to evolve independently.

Additionally, there exists integration between the AuctionService's Auction Scheduler and the ItemCatalogue service. The AuctionService features an auction scheduler set to execute every 3 minutes, actively engaging with the ItemCatalogue service by calling the getAuctionedItems() endpoint. Following this interaction, the AuctionService handles the obtained list of auctioned items, dynamically aligning their start and end times with the current time. This dynamic synchronization functions as a timer, initiating or concluding auctions as required.

Dependency injection is employed for injecting service implementations into controllers, promoting loose coupling and easing testing and maintenance. Controllers review service instances through constructor injection, contributing to a modular and flexible system.

Finally, the introduction of the QueryResult class for standardized responses improves consistency in communication across microservices. This simplifies error handling, making it easier for clients to interpret and process responses from different services.

Our E-Auction system thrives on microservices architecture, ensuring scalability and maintainability. The client-server model enables independent development and scaling. The Common Library, implemented as singletons, ensures consistency with minimal resource usage.Integration patterns, like the Observer Pattern and dynamic synchronization in the AuctionService's Scheduler, foster modularity. Dependency injection in controllers enhances system flexibility. The QueryResult class improves communication consistency.In essence, these choices create a robust and scalable E-Auction system, meeting criteria for scalability, maintainability, and efficient microservices communication.

SEQUENCE DIAGRAMS

USE CASE 1:

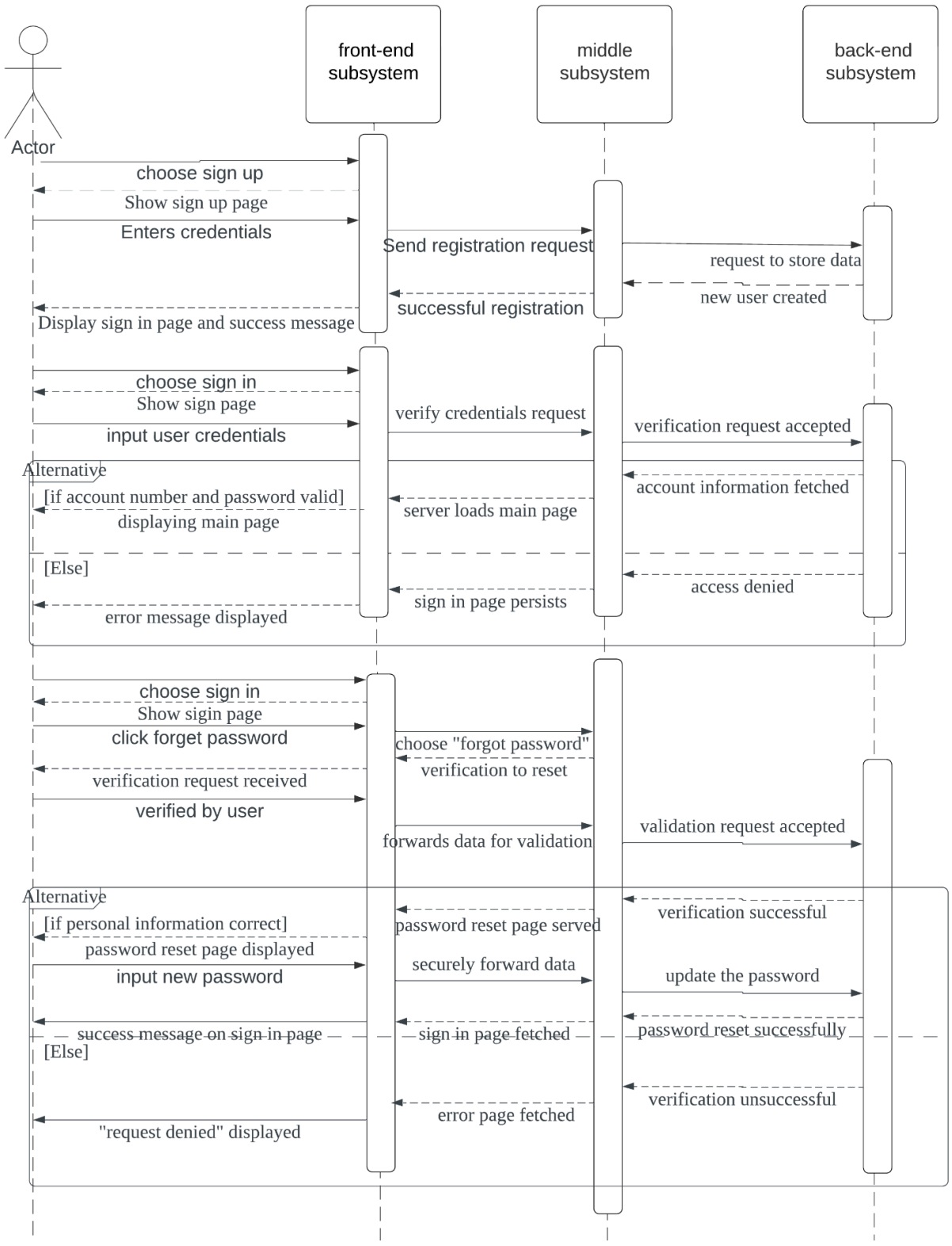

Figure 1.1 ***Sequence diagram for use case 1***

Illustrates a site visitor signing in, creating an account and resetting their password

USE CASE 2:

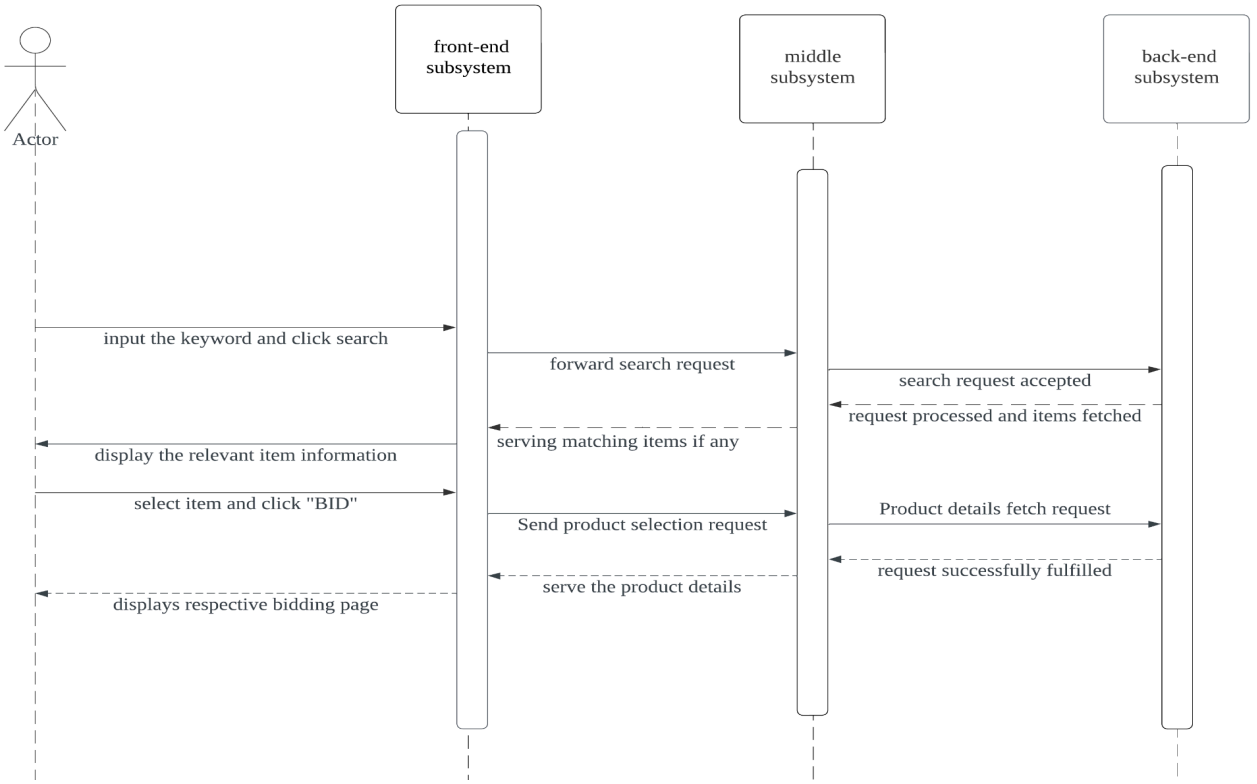

Figure 1.2: ***Sequence diagram for Use case 2***

Illustrates when a site visitor signs in, searches for an item, information is returned and the visitor submits a bid

USE CASE 3:

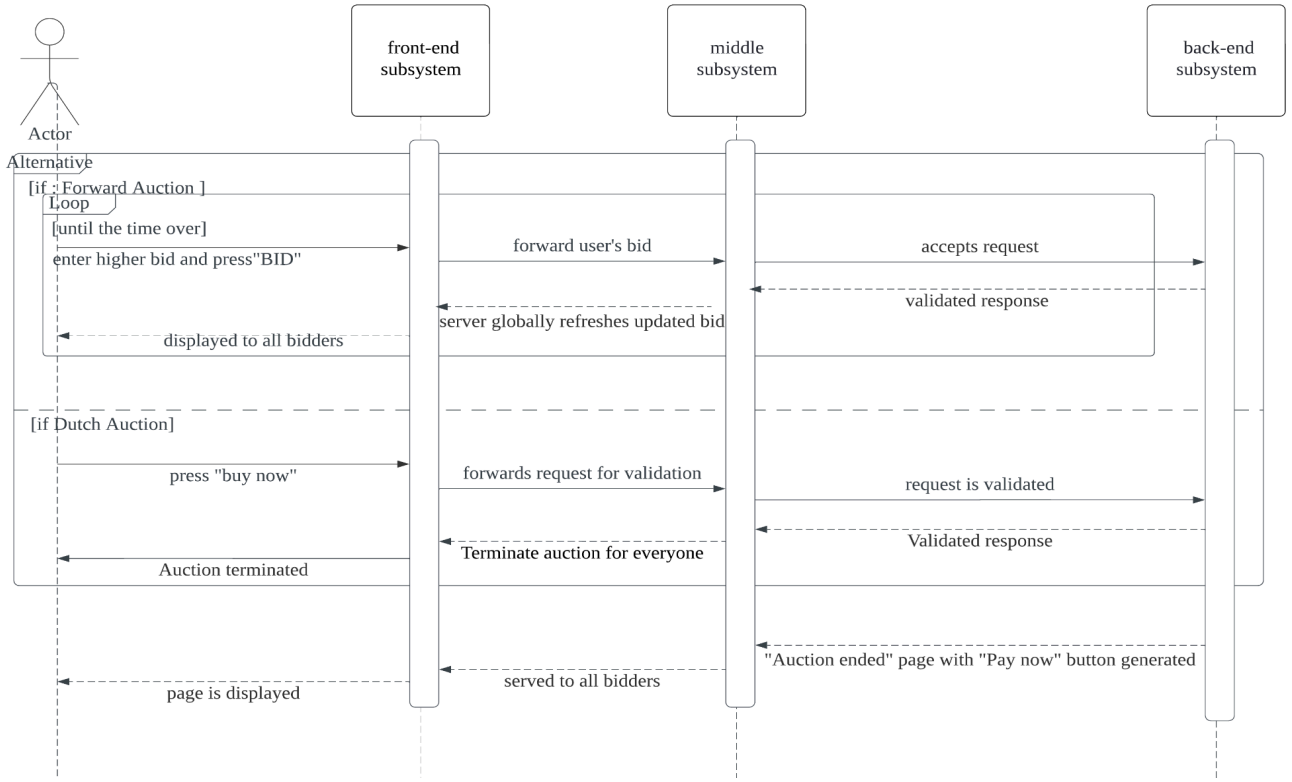

Figure 1.3: ***Sequence diagram for Use case 3***

Illustrates when a site visitor selects an item for bid apart of a forward auction, and when a site visitor selects an item for bid apart of a dutch auction

USE CASE 4:

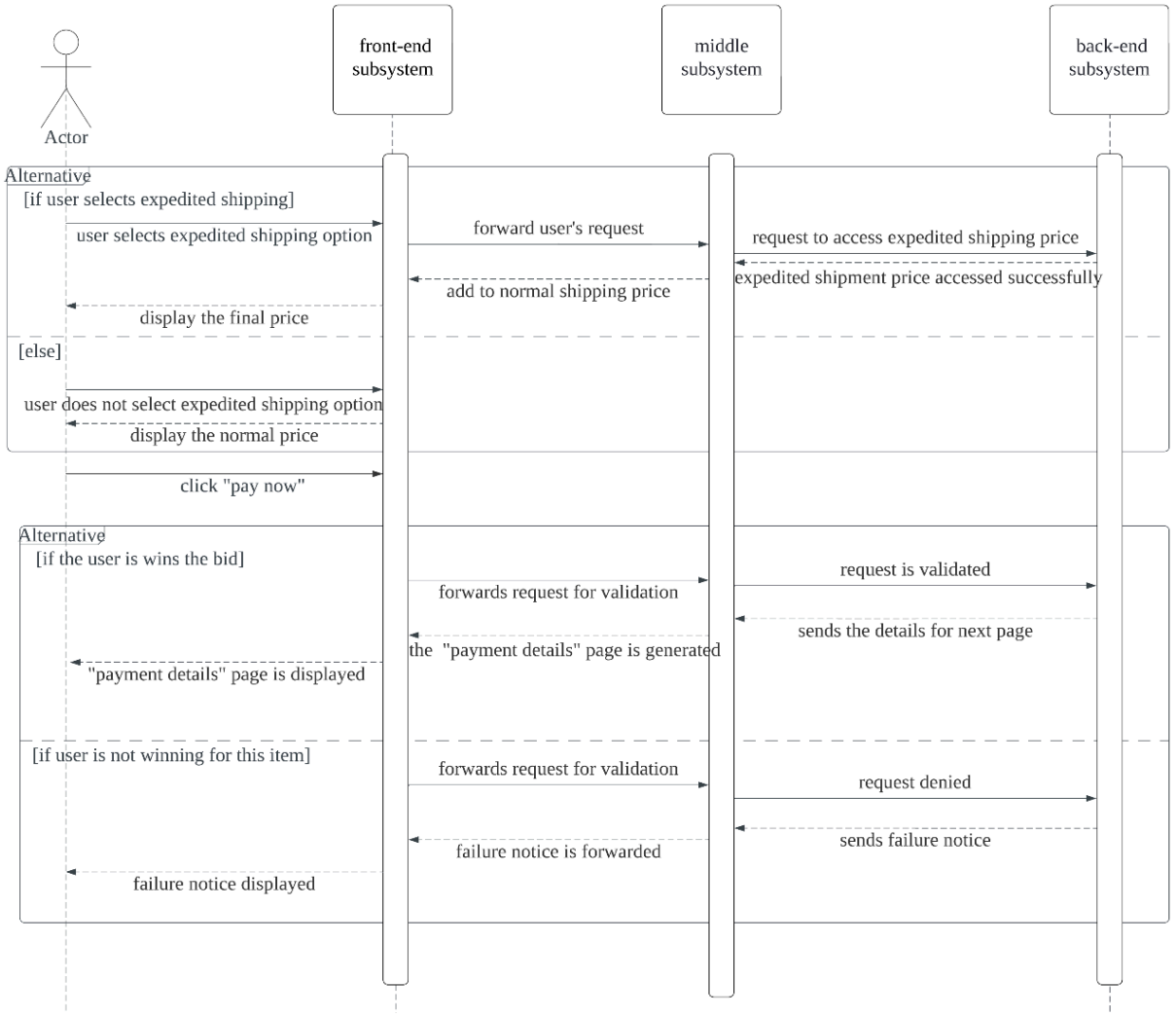

Figure 1.4: ***Sequence diagram for Use case 4***

Illustrates when a user wins or loses a forward auction, and when a user wins or loses a dutch auction

USE CASE 5:

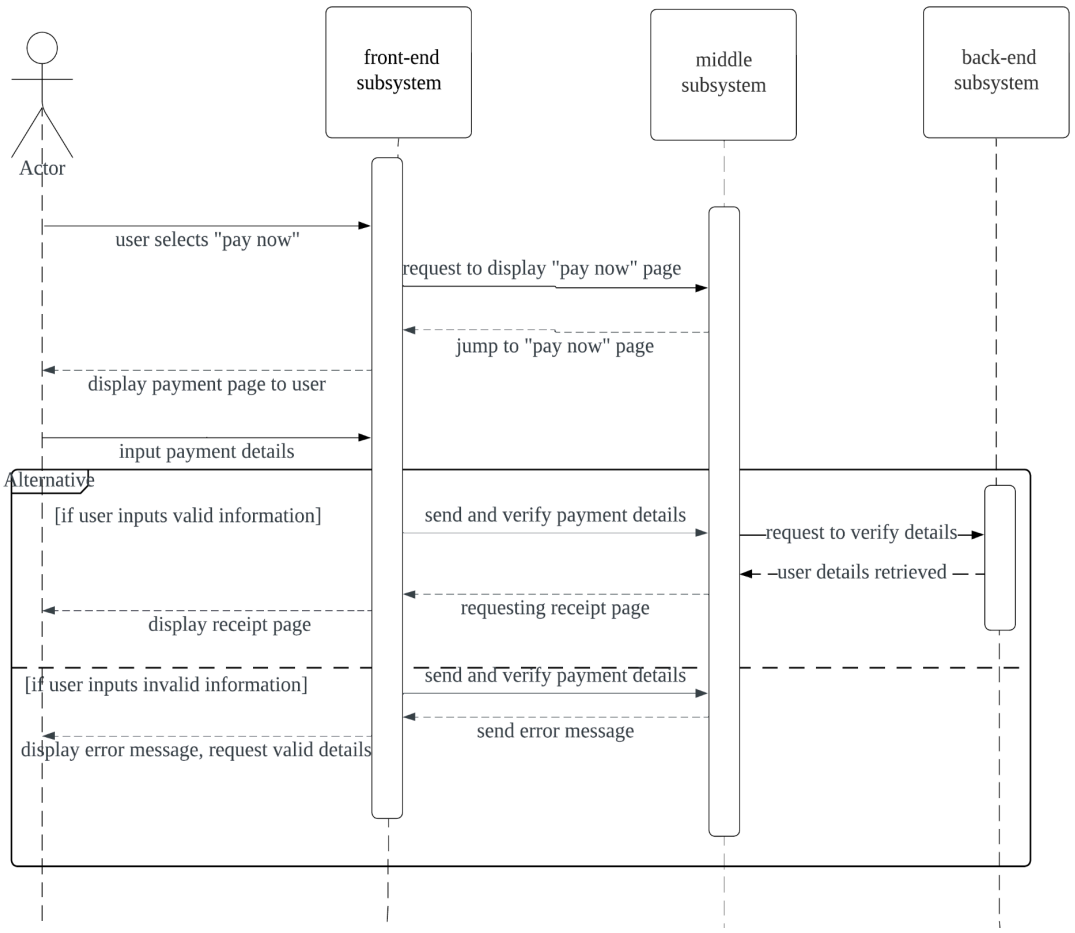

Figure 1.5: ***Sequence diagram for Use case 5***
Illustrates the payment process for an auction winner

USE CASE 6:

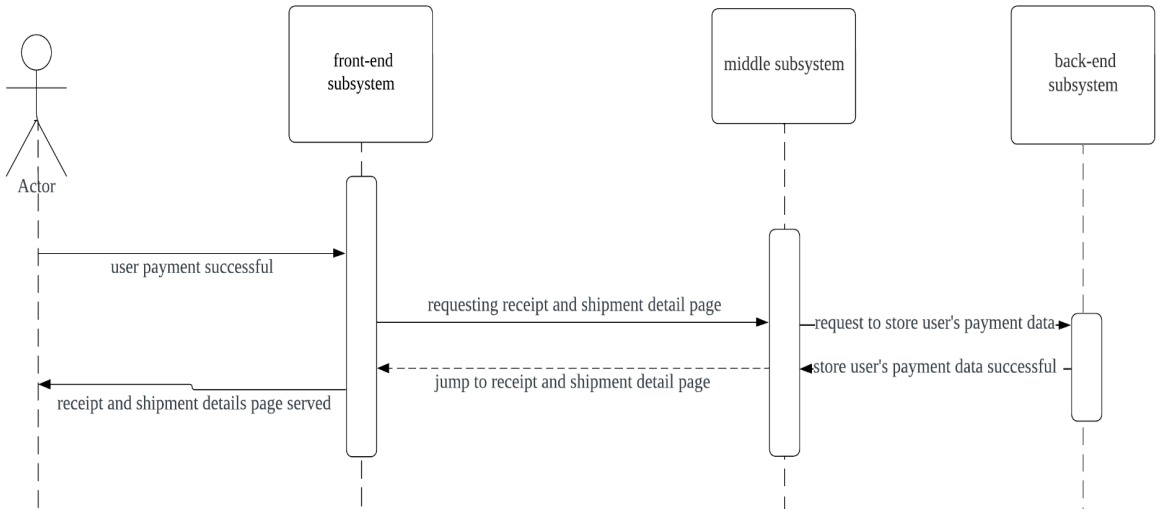

Figure 1.6: ***Sequence diagram for Use case 6***
Illustrates when payment is a success, shipment details

USE CASE 7:

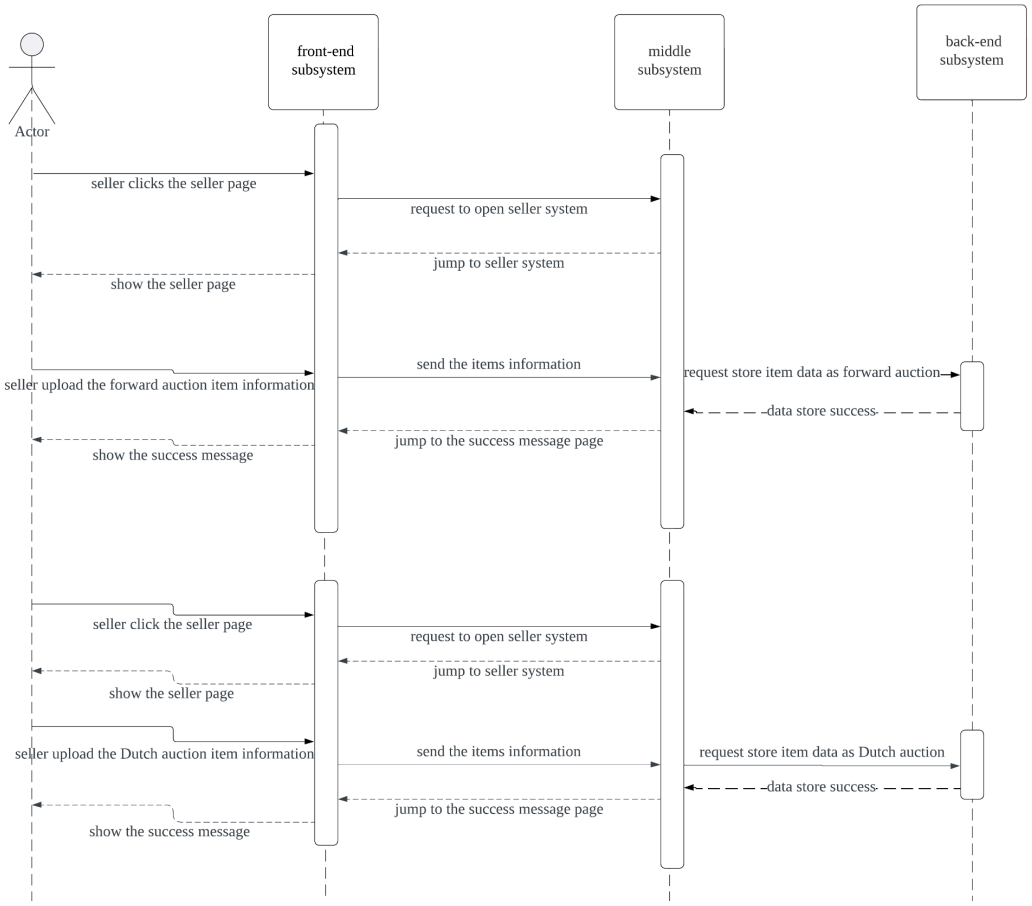

Figure 1.7: ***Sequence diagram for Use case 7***
Illustrates seller creating forward auction or dutch auction

USE CASE 8:

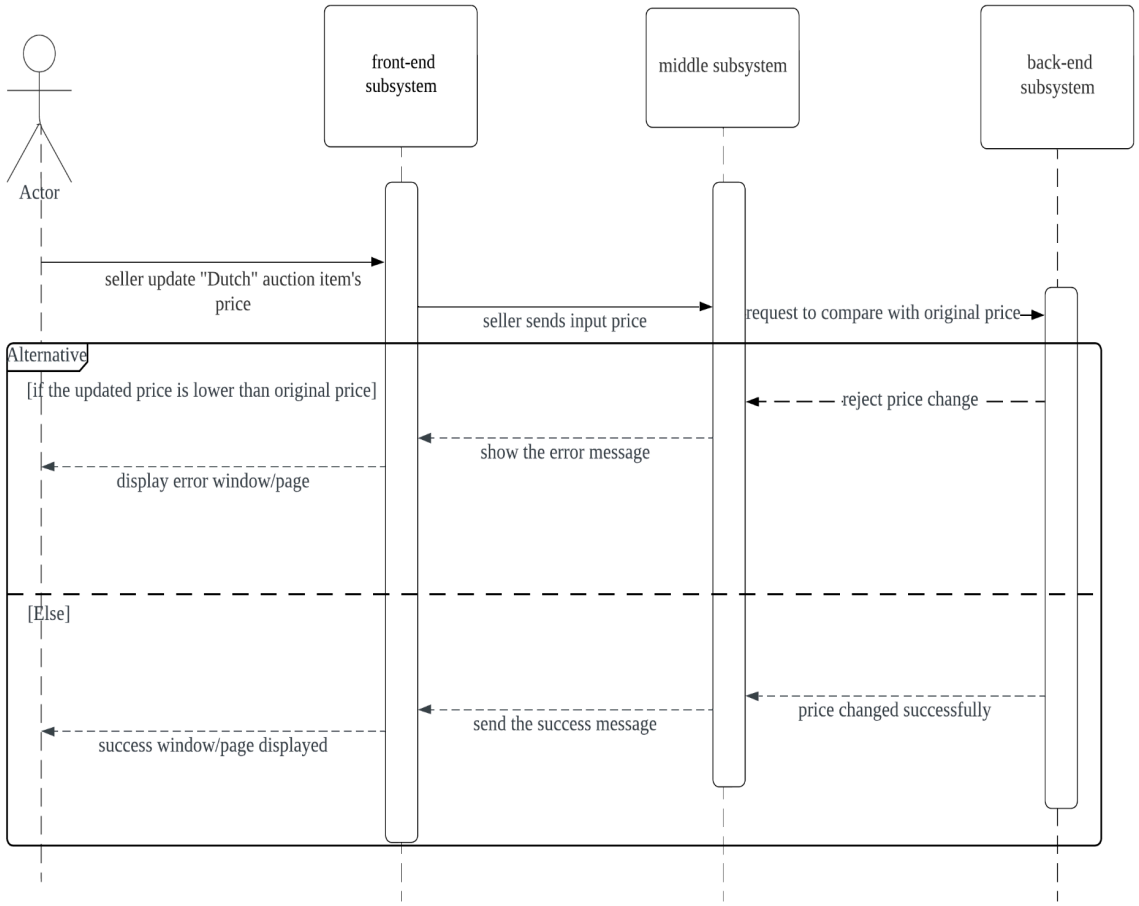

Figure 1.8: ***Sequence diagram for Use case 8***
Illustrates a seller updating the price of an item for a dutch style auction

ACTIVITY DIAGRAMS

USE CASE 1:

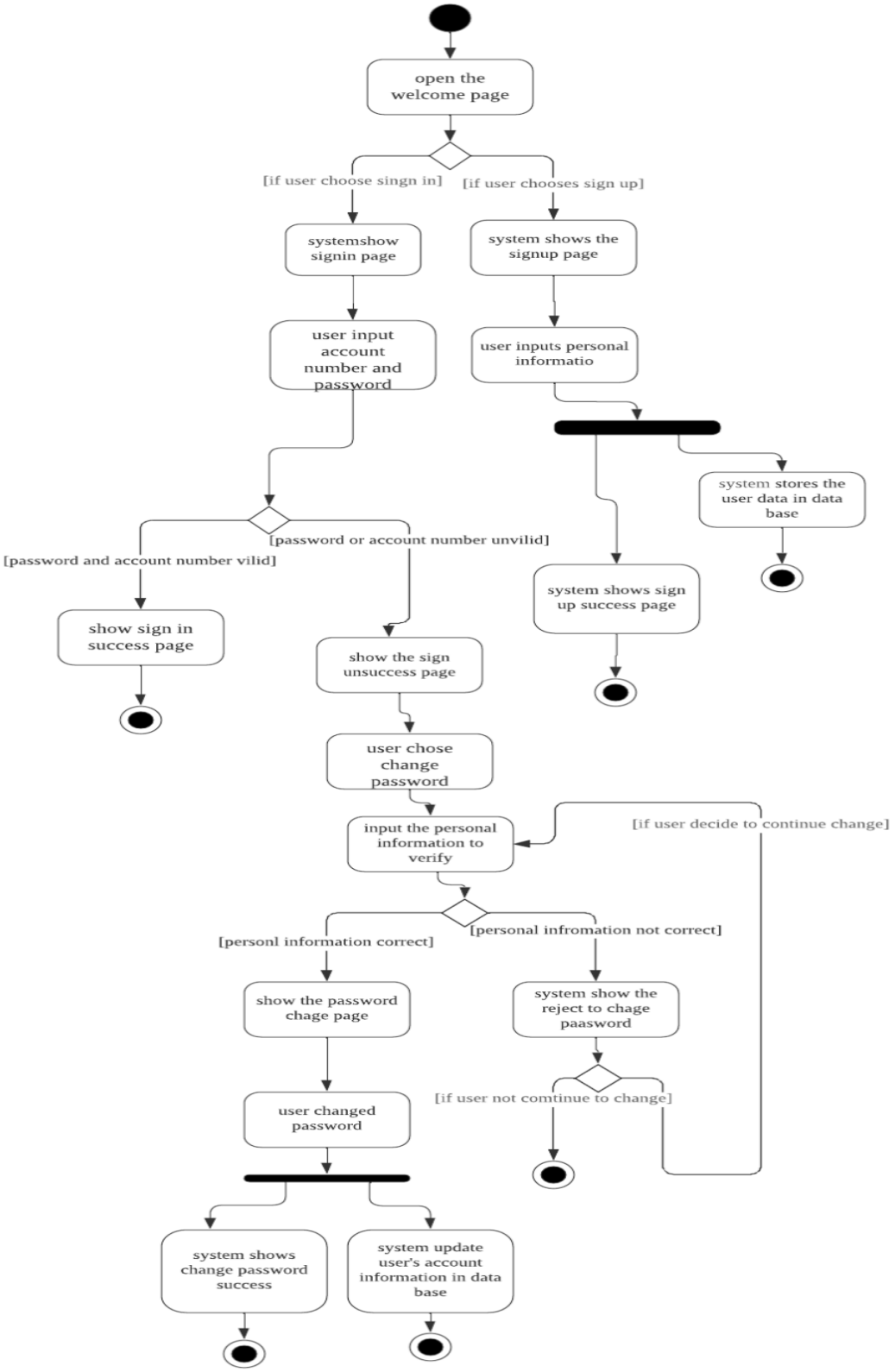

Figure 2.1: ***Activity diagram for use case 1***
Illustrates when the user signs in or creates an account, and if forgot password is selected

USE CASE 2:

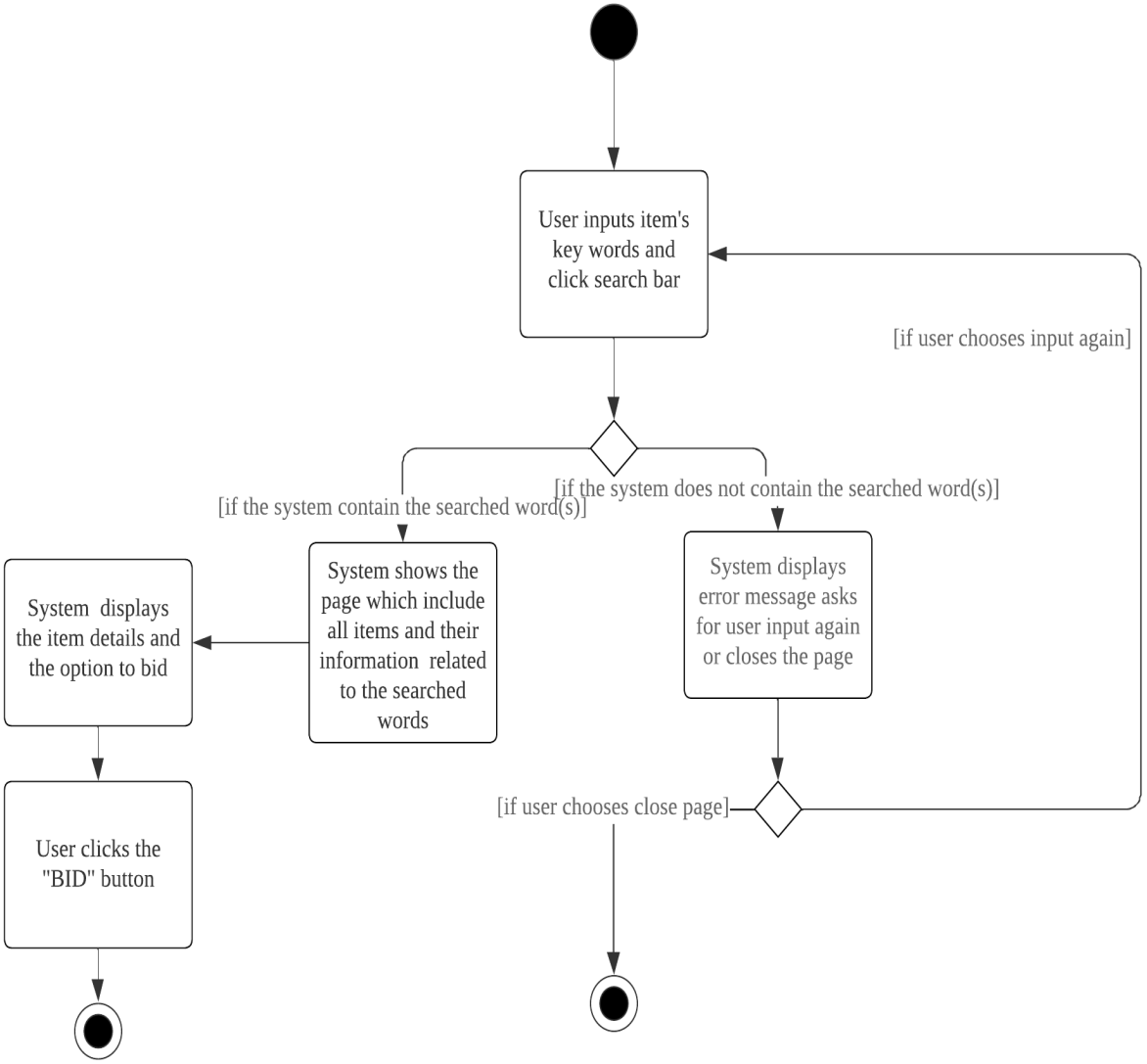

Figure 2.2: ***Activity diagram for use case 2***

Illustrates when a user has successfully signed in and searches for an item if the item exists to display the item, if not, an error is displayed

USE CASE 3:

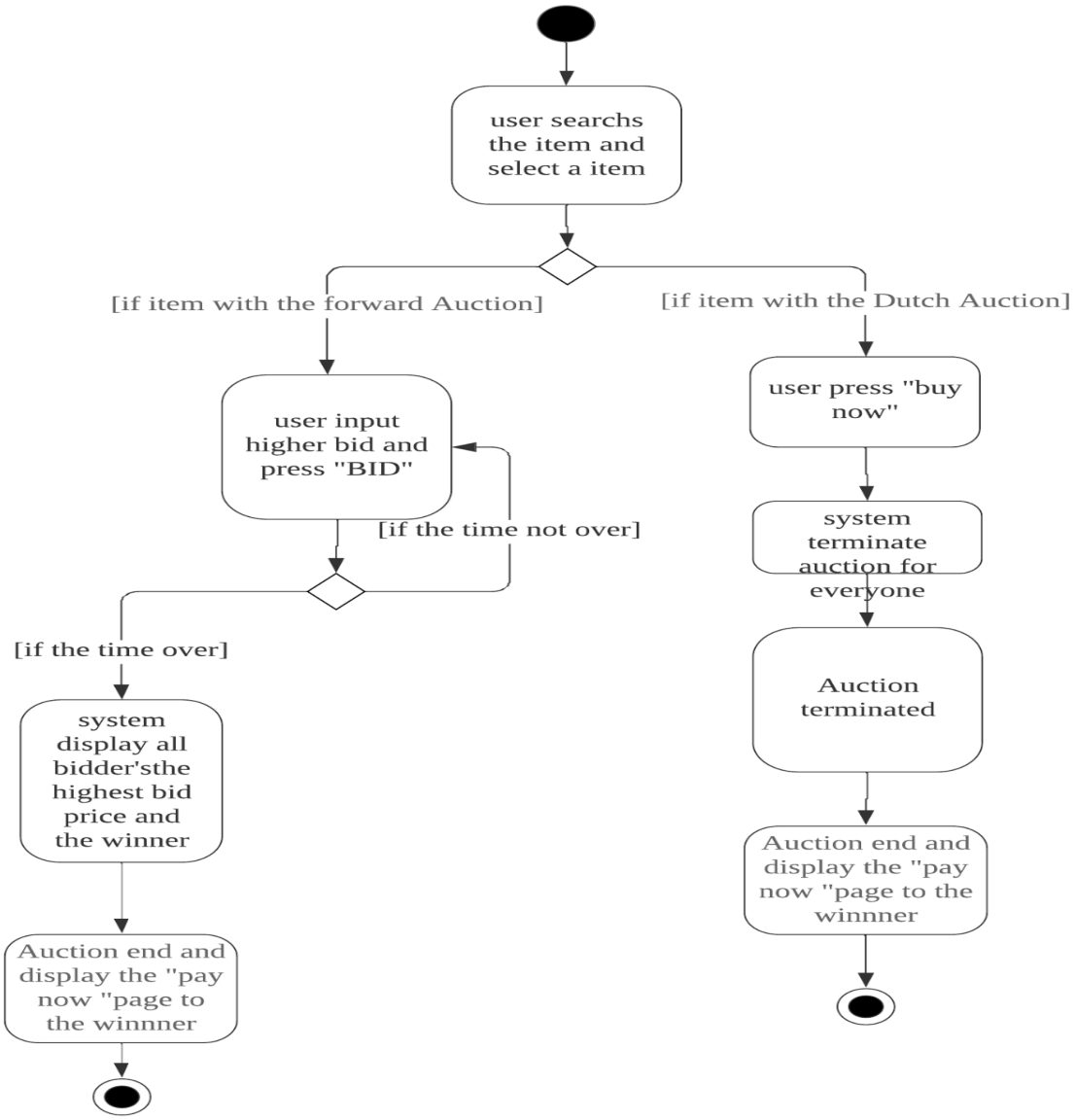

Figure 2.3: ***Activity diagram for use case 3***

Illustrates user successfully searching for an item and bidding if forward auction and buying if dutch auction

USE CASE 4:

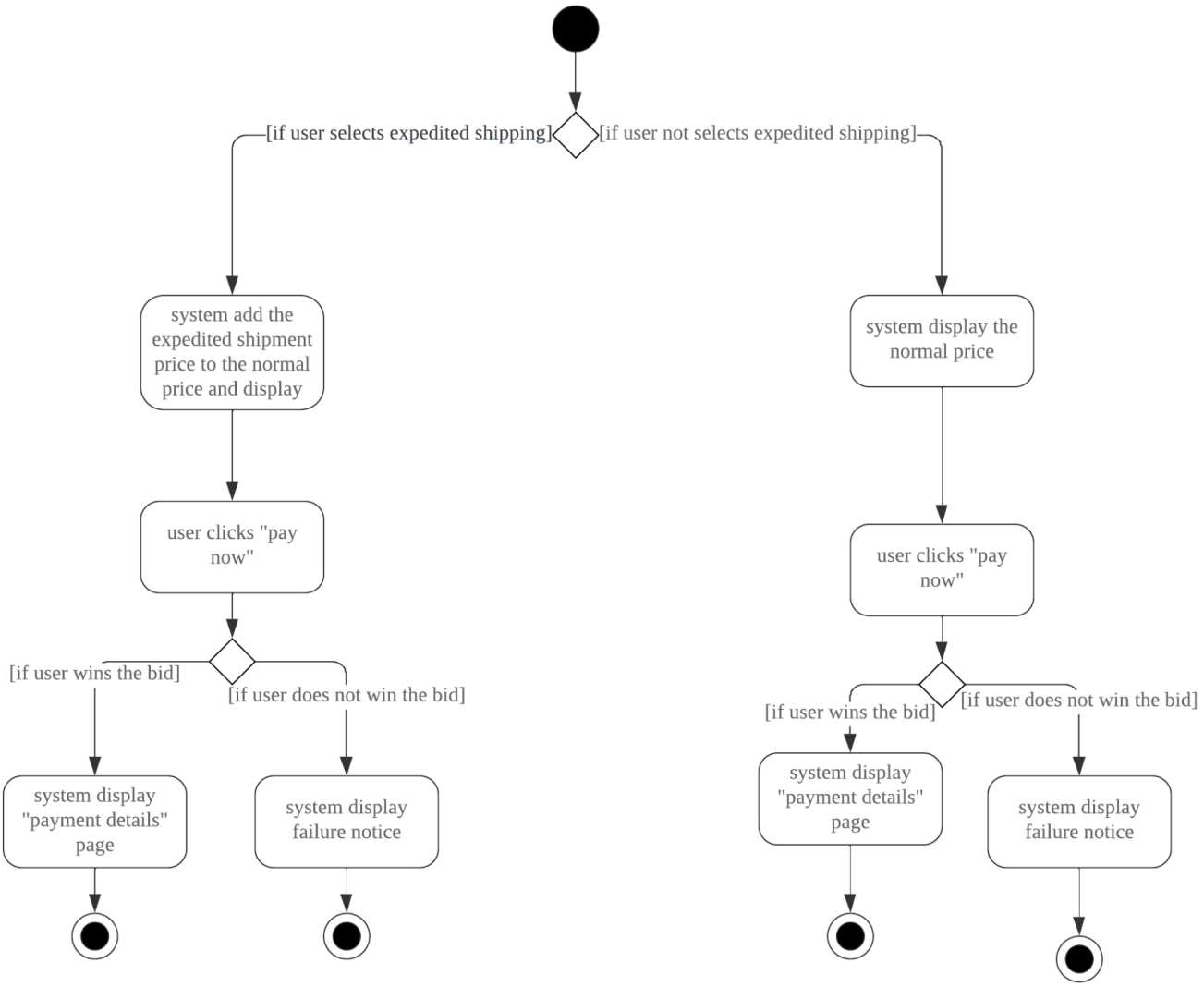

Figure 2.4: ***Activity diagram for use case 4***
Illustrates an auction ending and a user winning if they are the highest bidder in a forward auction

ARCHITECTURE

The architectural overview depicted in the diagram below is a comprehensive representation of our E-Auction system, featuring distinct elements in the frontend, middle-end, and backend. Starting with the front-end, implemented in React, it resides on the leftmost section of the diagram and is responsible for rendering user interfaces, facilitating event-driven communication through the Pub-Sub architecture, and managing HTTP/HTTPS requests and responses with the server side.

In the central portion, the middle-end is represented by various services implemented in Spring Boot with Java, each following a layered architecture. The AuthorizationService ensures secure user authentication and authorization, while the PaymentService handles secure payment transactions. The ItemCatalogueService manages the auction item catalog, the BiddingService facilitates bidding processes, the AuctionService orchestrates auction-related operations, the OrderProcessingService handles post-payment processes, and the ShippingService manages shipping logistics. These services adhere to the Model-View-Controller (MVC) principles, where controllers route requests, services handle business logic, and models represent data structures. Additionally, the Common Library shared among these services ensures a consistent representation of shared classes, promoting code reuse and minimizing redundancy.

On the right side, the Data Access Layer, implemented in Java with JDBC, interacts directly with the database. This layer is responsible for executing CRUD operations on the database, ensuring seamless communication between the backend services and the underlying database

This architecture establishes a clear distinction between frontend and backend components, showcasing their interactions and responsibilities. The React frontend handles user interfaces, event publishing, and subscription, while the Spring Boot backend manages various services with well-defined layers. The inclusion of the Common Library fosters consistency across microservices.

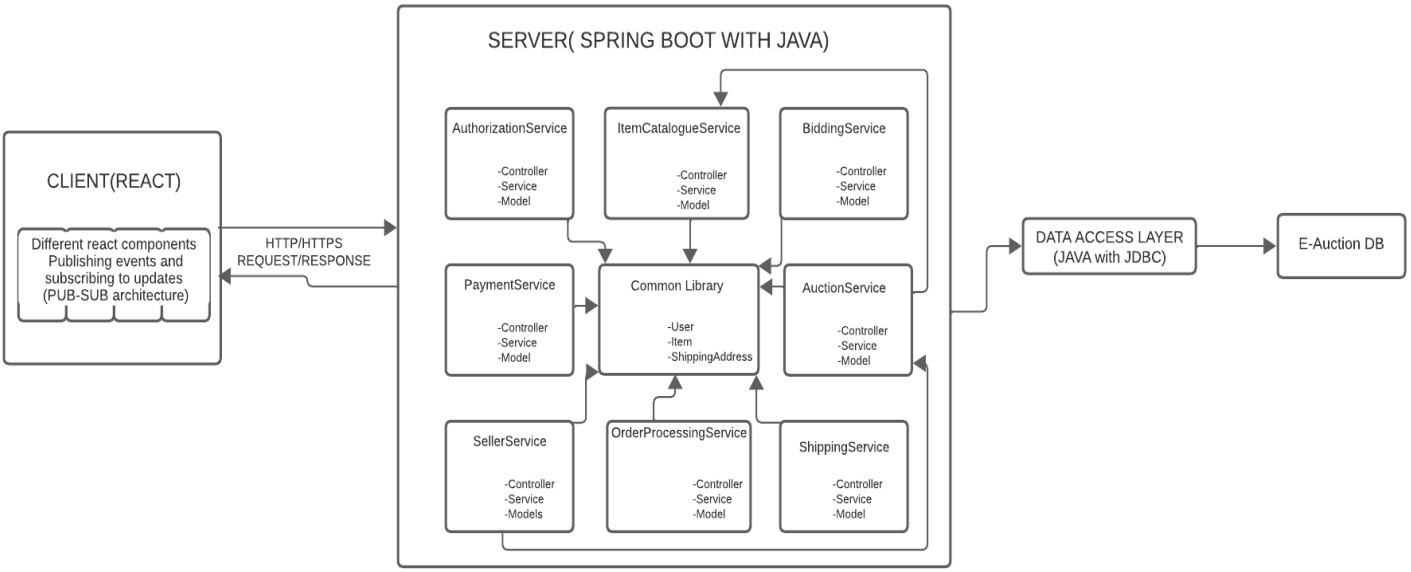

Figure 3: ***Bidding System's architectural overview***

The figure above shows the architectural overview of our bidding system. It has a frontend built with react using the pub-sub architecture within the frontend components. We have a server that communicates with the front end using http/https requests and responses. In the backend, we implement API's (or services) adhering to microservices architecture with each service incorporating Model-View-Controller(MVC) principles, using Java with Spring Boot as the framework.Notably, a Common Library serves as a centralized repository for shared classes, enhancing consistency across microservices. There is one databases that interact with the backend via a data access layer using JDBC. The frontend acts as a client to the backend while the backend acts as both a server to the frontend and a client to the database and the database acts as a server to the backend.

MODULES			
Module Name	**Description**	**Exposed Interface**	**Interface Description**
AuthorizationService	Handles the tasks of allowing users to sign in, sign up for new accounts, handles an extra layer of security through two-factor authentication, and manages user-related data.	Authorization	Provides the functionality of creating new accounts, signing into an existing account, enabling two-factor authentication, and resetting passwords.
ItemCatalogueService	Manages information and data related to auction items.	ItemCatalogue	Provides functionality and data or information related to items.
BiddingService	Manages data and functionality related to bidding.	Bidding	Provides functionality of placing bids and updating the bidding prices.
AuctionService	Manages the auctions.	Auction	Provides functionalities to begin and end the auction and get updates on the remaining time for the auction.
PaymentService	Manages the payment process.	Payment	Provides payment functionality.
OrderProcessingService	Manages the post-payment process including generating a receipt.	OrderProcessing	Provides functionality for generating receipts and updating the inventory for the auctioned item.
ShippingService	Handles the shipment process.	Shipping	Provides the functionality of calculating the shipping address and printing the shipping details.
SellerService	Handles operations related to a seller.	Seller	Provides the functionality of adding new items to the item catalogue and decreasing the bidding during Dutch auctions.

Table 1.1: ***Modules Overview***

The table presented below provides a comprehensive overview of the modules and their exposed interfaces.

Interfaces		
Interface Name	**Operations**	**Operation Descriptions**

Authorization	AuthorizationQueryResult : signUp(User user, String accountType) used by Front-end AuthorizationQueryResult : signIn(String userName, String password) used by Front-end AuthorizationQueryResult : passwordReset(String username, String newPassword) used by Front-end AuthorizationQueryResult : getUserDetails(int userId) used by AuctionService	signUp(User user, String accountType): registers a new user. signIn(String userName, String password): lets a registered user login to the application. passwordReset(String username, String newPassword): resets a user's password. getUserDetails(int userId): fetches the user details with the given Id
ItemCatalogue	ItemCatalogueQueryResult : search (String keyword) used by Front-end ItemCatalogueQueryResult : getAuctionedItems() used by Front-end and AuctionService ItemCatalogueQueryResult : getItem(int itemId) used Front-end	search (String keyword): Gets a list of items based on the search criteria. getAuctionedItems(): Gets the list of items up for auction. getItem(int itemId): Gets detailed information about a specific item
Bidding	BiddingQueryResult: placeBid (int itemId, double newPrice, String userName) used by Front-end	placeBid (int itemId, double newPrice, String userName): Places a bid for an item in a forward auction.
Auction	AuctionQueryResult: startAuction(int itemId) used by Front-end AuctionQueryResult: getRemainingTimeUpdate(int itemId) used by Front-end AuctionQueryResult: endAuction (int itemId) used by Front-end AuctionQueryResult: createAuction(Map<String, String> auction) used by SellerService	startAuction(int itemId): Starts an auction. getRemainingTimeUpdate(int itemId): Get updates on the time remaining for an auction. endAuction (int itemId): Determines the winner and ends an auction. createAuction(int itemId): Creates an auction for the given Item.
Payment	PaymentQueryResult: processPayment(String userName, int itemId) used by Front-end	processPayment(String userName, int itemId): Process a payment transaction.
OrderProcessing	OrderProcessingQueryResult: generateReceipt (int itemId, String userName) used by Front-end OrderProcessingQueryResult: updateItems(int itemId) used by Front-end	generateReceipt (int itemId, String userName): Generates a receipt after the payment is complete. updateItems(int itemId): Updates the inventory after an auction ends.

Shipping	ShippingQueryResult: calculateShippingCost(int itemId) used by Front-end ShippingQueryResult: setExpeditedShipping(int itemId) used by Front-end ShippingQueryResult: displayShippingDetails(int itemId) used by Front-end	calculateShippingCost(int itemId): Calculates shipping cost based on its type. setExpeditedShipping(int itemId): Sets the shipment type to expedited shipment. displayShippingDetails(int itemId): Arrange the shipment of an item and return the expected shipment date.
Seller	SellerQueryResult: addSellItems(Item item) used by Front-end SellerQueryResult: updateDutchAuctionprice(int itemId, double newprice) used by Front-end	addSellItems(Item item): Adds an item to sell and updates its details. updateDutchAuctionprice(int itemId, double newprice): Decreases the price for a Dutch auction

Table 1.2: ***Interface Operations Details***

The table presented below provides details on the exposed interfaces and their operations.

ACTIVITIES PLAN

User Story	Priority
As a user, I want to be able to bid on items	13
As a user, I want to be able to pay for items on won auctions	7
As a user, I can sign up and create an account	9
As a user, I can sign in with my credentials	9
As a seller, the auctions must end correctly	13
As a user, I want to be able to select an item to bid for	9
As a user, after paying I want to be able to see the receipt and shipment details	9
As a user, I want to see a list of auctioned items	7
As a user, I can search for items	7
As a user, I can reset my password if forgotten	5

Table 2: Product Backlog
Table above illustrates the sprint backlog

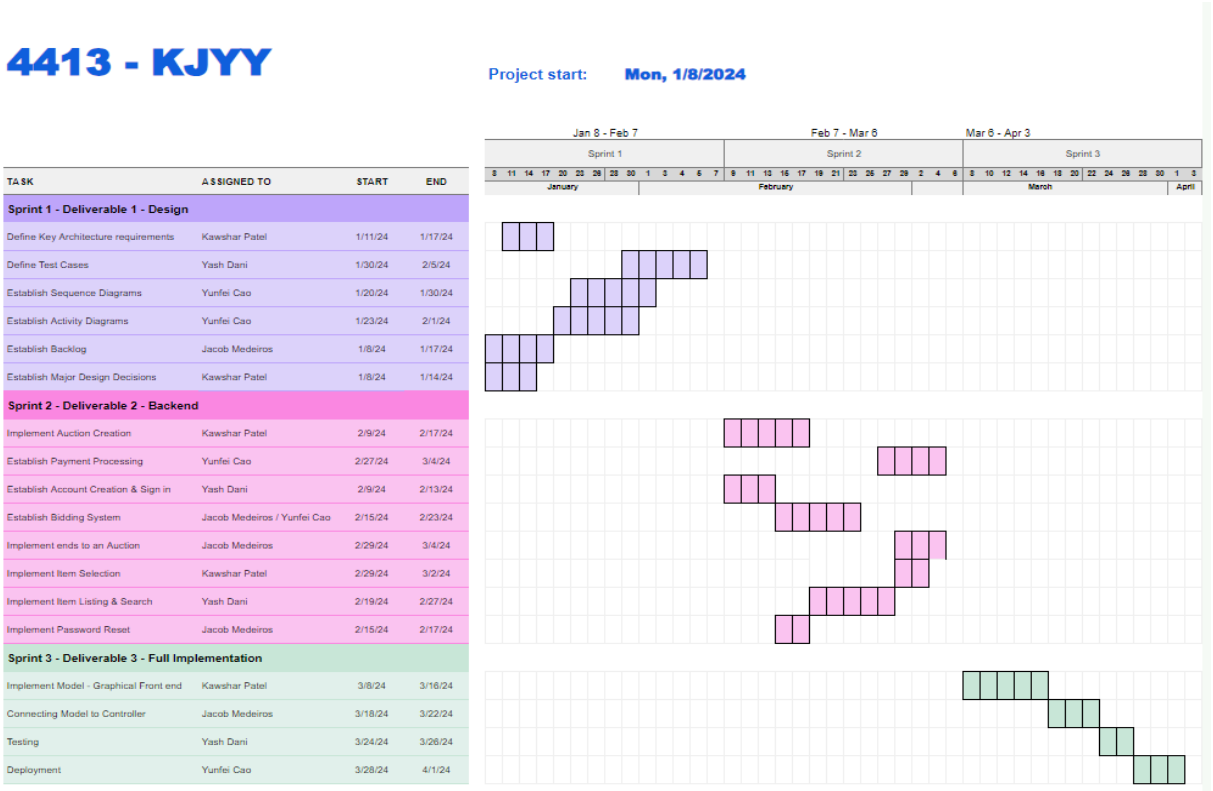

Figure 4: *Sprint Backlog*
Figure above illustrates the sprint backlog

GROUP MEETING LOGS

Present Group Members	Meeting Date	Issues Discussed / Resolved
Kawshar, Jacob, Yash, Yunfei	02/10	Duties divided
Kawshar, Jacob, Yash, Yunfei	02/17	Discussed shell front-end implementation & database
Kawshar, Jacob, Yash, Yunfei	02/29	Discussed backend implementation & linking to database
Kawshar, Jacob, Yunfei	03/04	Discussed current code & testing needs
Kawshar, Jacob, Yunfei	03/05	Discussed testing & code improvements
Kawshar, Jacob, Yunfei	03/06	Discussed testing & code improvements
Jacob, Yunfei	03/07	Discussed testing & code improvements
Jacob, Yunfei	03/08	Discussed testing & code improvements
Jacob, Yunfei	03/10	Final Review of Sprint 2

Table 3: ***Group meeting logs table***

TEST DRIVEN DEVELOPMENT

Test ID	TC001
Category	Evaluation of new user details registered on file/DB
Requirements Coverage	UC1.1-Successful-User-SignUp-Flow
Initial Condition	System has been initiated and runs and the user is presented with the welcome screen with Sign-up/Sign-in options
Procedure	1. User selects the sign-up option 2. User provides a valid user name, a password, their first name, their last name and their shipping address (street name, street number, city, country, postal code). 3. User selects the submit option to submit the above details
Expected Outcome	User is successfully registered, and the system navigates to the Sign-In page.
Notes	- (Alternate Flow) Failed Sign Up due to invalid credentials Future considerations: - A 2 Factor Authentication service could be implemented - A “Strong password” check could be implemented - Fails when the same user tries to sign up again

Table 6.1: Test Case TC001
This test outlines testing for the user sign up flow

Test ID	TC002
Category	Server verifies new user details before registering on file/DB
Requirements Coverage	UC1.1-Unsuccessful-User-SignUp-Flow
Initial Condition	System has been initiated and runs and the user is presented with the welcome screen with Sign-up/Sign-in options
Procedure	1. User selects the sign-up option 2. User provides a username, a password, their first name, their last name and their shipping address (street name, street number, city, country, postal code) but at least one of the fields is invalid or blank 3. User selects the submit option to submit the above details
Expected Outcome	User receives an error message and the Sign-Up page persists.
Notes	

Table 6.2: Test Case TC002
This test outlines testing for unsuccessful user sign-up

Test ID	TC003
Category	Server verifies user details registered on file/DB
Requirements Coverage	UC1.2-Successful-User-Login-Flow
Initial Condition	System has been initiated and runs and a registered user is presented with the welcome screen with Sign-up/Sign-in options
Procedure	1. User selects the sign-in option 2. User enters a valid username and password 3. User selects the submit option to submit the above details
Expected Outcome	User is successfully signed in and the system navigates to the main page.
Notes	- Implementing a forgot password mechanism - (Alternate flow) Failed Sign in due to invalid credentials Future considerations: - "Remember Me" option for convenience

Table 6.3: Test Case TC003

This test outlines testing for successful user login

Test ID	TC004
Category	Server verifies user details registered on file/DB
Requirements Coverage	UC1.2-Unsuccessful-User-Login-Flow
Initial Condition	System has been initiated and runs and a registered user is presented with the welcome screen with Sign-up/Sign-in options
Procedure	1. User selects the sign-in option 2. User enters an invalid username and password 3. User selects the submit option to submit the above details
Expected Outcome	User receives an error message and the Sign-In page persists
Notes	

Table 6.4: Test Case TC004

This test outlines testing for unsuccessful user login

Test ID	TC005
Category	Retrieval of matching items from file/DB
Requirements Coverage	UC2.1-Successful-Item-Search-Flow
Initial Condition	User is signed in, and the system is displaying the main page.
Procedure	1. User enters a keyword in the search bar. 2. User selects the "search" option
Expected Outcome	System displays a list of items (up for auction or currently being auctioned) matching the search keyword, retrieved from the file/DB
Notes	- (Alternate Flow) No items matching search results Future considerations: - Might implement advanced search filters for more precise results.

Table 6.5: Test Case TC005

This test outlines testing for successful item search

Test ID	TC006
Category	Retrieval of matching items from file/DB
Requirements Coverage	UC2.1--Item-Search-No-Matching-Results-Flow
Initial Condition	User is signed in, and the system is displaying the main page.
Procedure	1. User enters a keyword in the search bar. 2. User selects the “search” option
Expected Outcome	System displays a message indicating no matching items found
Notes	

Table 6.6: Test Case TC006
This test outlines testing for item search with no matching results

Test ID	TC007
Category	Matching auctioned items retrieved from file/DB are served to the user
Requirements Coverage	UC2.2-Successfully-Display-Auctioned-Items
Initial Condition	User is signed in, and the system is displaying the list of all matching auctioned items after searching keyword
Procedure	1. User views or scrolls through the displayed list of matching auctioned items
Expected Outcome	System displays the full item name, the current bidding price, the type of auction, and the remaining time (applicable only for forward auctions) for each matching item.
Notes	

Tabke 6.7: Test Case TC007
This test outlines testing for displaying auctioned items

Test ID	TC008
Category	Selected item details are retrieved from file/DB is served to the user
Requirements Coverage	UC2.3-Successful-Item-Selection
Initial Condition	User is viewing the displayed items, and their details and there is a selection radio button for each displayed item which allows the user to select the item to bid for
Procedure	1. User clicks the radio button belonging to the desired item 2. User selects the “Bid” option.
Expected Outcome	System displays the item details and the option to bid
Notes	- User can select only one item to bid for - We assume a bidder bids for only one item for each log-in session they have initiated Future Considerations: - User may have multiple log-in sessions in parallel in different browsers

Table 6.8: Test Case TC008
This test outlines testing for item selection

Test ID	TC009
Category	Evaluation of server for refreshing the page after every bid and fetching data accurately from the respective file/DB
Requirements Coverage	UC3.1-Successful-Forward-Auction-Bidding-Flow
Initial Condition	User is on the bidding screen for the selected forward auction item where item details and option to bid are displayed
Procedure	1. User provides a new bidding price which must be higher than the current price 2. User selects the “Bid” option.
Expected Outcome	Once a bid is submitted, the system (server) refreshes the page and the new highest bidding price and the highest bidder are displayed to all users bidding for this item so that the current highest price and the ID of the current highest bidder are always displayed
Notes	- The auction ends when the time expires and users are directed to “Auction ends” page and served with a “pay now” button Future Considerations: - Real-Time bidding updates feature

Table 6.9: Test Case TC009

This test outlines testing for forward auction bidding

Test ID	TC010
Category	Evaluation of server directing all users to “auction ends” page.
Requirements Coverage	UC3.2-Successful-Dutch-Auction-Bidding-Flow
Initial Condition	User is on the bidding screen for the selected dutch auction item where item details and the option to bid are displayed
Procedure	1. User clicks on the "Buy Now" button.
Expected Outcome	- System immediately terminates the auction and the user is directed to “Auction ends” page and served with a “pay now” button.
Notes	Future Considerations: - Countdown timer for dutch auctions.

Table 6.10: Test Case TC010

This test outlines testing for dutch auction bidding

Test ID	TC011
Category	Evaluation of server directing the user to the appropriate page and retrieving details from the DB
Requirements Coverage	UC4-Successful-Auction-Ended-Pay-Now-Flow
Initial Condition	Auction has ended (either time ended for forward auction or because of a “Buy Now” in Dutch auctions) and the users are served a “Auction Ended” page with a “Pay Now” button. Here, consider a winning user (“successful flow”)
Procedure	1. (Optional) User selects the “expedited shipment” option 2. User clicks on the "Pay Now" button
Expected Outcome	- System displays the payment page with the user and shipping details from the DB that stores data when the user signed up
Notes	- If the user selects the “expedited shipment” option, then the price of expedited shipment is added to the normal shipping price. This expedited shipment cost information can be included in the Catalogue DB for each specific item - (Alternate flow) The user is not a winner for the specific item and is served with a failure notice upon clicking the “pay now” button

Figure 6.11: Test Case TC011
This test outlines testing for auction ending

Test ID	TC012
Category	Evaluation of server directing the user to the appropriate page and retrieving details from the DB
Requirements Coverage	UC4-Unsuccessful-Auction-Ended-Pay-Now-Flow
Initial Condition	Auction has ended (either time ended for forward auction or because of a “Buy Now” in Dutch auctions) and the users are served an “Auction Ended” page with a “Pay Now” button. Here, consider a losing user (“unsuccessful flow”)
Procedure	1. (Optional) User selects the “expedited shipment” option 2. User clicks on the "Pay Now" button
Expected Outcome	- System displays an error message indicating that the user cannot pay for a non-winning item
Notes	

Table 6.12: Test Case TC012
This test outlines testing for unsuccessful auction end

Test ID	TC013
Category	Evaluation of server's payment processing and operations on multiple respective DBs. (Confirm names)
Requirements Coverage	UC5-Successful-Payment-Flow
Initial Condition	User is on the Payment page displaying the user/shipping details from the DB that stores data when the user signed up
Procedure	1. User adds valid payment details, i.e., the credit card number, the name on the card, the expiration date and the security code 2. User clicks the "Submit button"
Expected Outcome	- System processes the payment, and the receipt page with shipping details is displayed
Notes	- The payment includes the item price plus the shipping cost. - (Alternate Flow) Payment declines due to unsupported pathway or invalid details Future Considerations: - Integrate multiple payment gateways.

Table 6.13: Test Case TC013

This test outlines testing for successful payment flow

Test ID	TC014
Category	Evaluation of server's payment processing and operations on multiple respective DBs. (Confirm names)
Requirements Coverage	UC5-Unsuccessful-Payment-Flow
Initial Condition	User is on the Payment page displaying the user/shipping details from the DB that stores data when the user signed up
Procedure	1. User adds invalid payment details, i.e., one of the credit card number, the name on the card, the expiration date or the security code is invalid or blank (eg: expired credit card) 2. User clicks "Submit button"
Expected Outcome	- System displays an error message indicating that the payment was unsuccessful and the payment page persists
Notes	

Table 6.14: Test Case TC014

This test outlines testing for unsuccessful payment

Test ID	TC015
Category	Evaluation of server's ability to fetch and display accurate details regarding a transaction from the DBs
Requirements Coverage	UC6-Receipt-Page-Shipment-Details-Display
Initial Condition	Payment is Successful
Procedure	1. User views the receipt and shipment details
Expected Outcome	- System displays the total amount paid as well as the shipping details
Notes	- Consider that each item has a separate shipping time - System does not need to calculate the actual shipping date from the current date; just indicate that "The Item will be shipped in xxx days" - Shipping time information can be included in the Catalogue DB for each specific item Future Considerations: - Sending email notifications and tracking details

Table 6.15: Test Case TC015

This test outlines testing for displaying shipment details

Test ID	TC016
Category	Evaluation that server stores new entry into DB
Requirements Coverage	UC7-Seller-Uploads-Item
Initial Condition	User is logged in as seller
Procedure	1. User (Seller) uploads all required information regarding the item such as the description, the type of auction (Dutch or forward), the duration of the auction and the starting bid price
Expected Outcome	- System adds the item to the Catalogue DB with relevant details
Notes	

Table 6.16: Test Case TC016

This test outlines testing for seller uploading an item

Test ID	TC017
Category	Evaluation that server updates an existing entry in the DB and broadcasts it to all engaged users
Requirements Coverage	UC7-Seller-Updates-Dutch-Auction-Price
Initial Condition	Seller is logged in and is active in an ongoing Dutch Auction
Procedure	1. Seller updates (decreases) the price for a Dutch auction item
Expected Outcome	- System updates the auction details with the new price and serves it to all the users engaging in that auction
Notes	

Table 6.17: Test Case TC017

This test outlines testing for a seller updating a dutch auction price