Process MeNtOR 1.0

E-Commerce Website (ECM)
Design Document

Version:	3.0
Print Date:	October 20, 2023
Release Date:	December 10, 2023
Release State:	Development
Approval State:	Submitted
Approved by:	Ruth, Kedamawi, Yaqub, Jay
Prepared by:	Ruth, Kedamawi, Yaqub, Jay
Reviewed by:	Ruth, Kedamawi, Yaqub, Jay
Path Name:	EECS 4413/ProjectReport
File Name:	ProjectReport
Document No:	1

object oriented pty. ltd.

Document Change Control

Version	Date	Authors	Summary of Changes
1.0	24/09/2023	Ruth, Kedamawi, Yaqub, Jay	Model View Controller & Protocols REST, HTTP
2.0	10/10/2023	Ruth, Kedamawi, Yaqub, Jay	Backend Server Database Model Added with Pub-Sub functionality
3.0	19/10/2023	Ruth, Kedamawi, Yaqub, Jay	Client Server architecture with RDBMS and front end UI interfaces

Document Sign-Off

Name (Position)	Signature	Date
Jay Sharma	*Jay Sharma*	Oct 20, 2023
Kedamawi Mengistu	*Kedamawi Mengistu*	Oct 20, 2023
Ruth Bezabeh	*Ruth Bezabeh*	Oct 20, 2023
Yaqub Ibrahim	*Yaqub Ibrahim*	Oct 20, 2023

Contents

1 Introduction

1.1 Purpose

The E-Commerce system described in this document is to create an auction-based e-commerce site that makes buying and selling goods and services through two different types of auctions: Forward Auctions and Dutch Auctions. This project aims to provide an implementation using frameworks, with a three-tier architecture consisting of a back-end, middle, and front-end subsystem.

1.2 Overview

This documentation details the process of developing our E-Commerce Website (ECM). Starting with the major decisions, we display our architect with sequence and activity diagrams and elaborate on our architecture with interface tables. Finally we have our meeting minutes and a list of test cases for our development when implementing the e-commerce website. Overall, we strived for sound architectural decisions that will enable through implementation and an efficient yet robust system.

1.3 Resources - References

- *Atlassian*
- *EECS 4413 Slides*
- *Draw.io for UML, Activity, Sequence Diagrams*
- *SDD Quality Assurance*
- *Jira*
- *EECS 4413 Project Description*
- *Eclipse Java Enterprise Edition 2023-06*

2 Major Design Decisions

Architectural Styles:

Client - Server - Our Ecommerce system is a web application that can be accessed through HTTP requests implemented using REST protocol. Users can connect to our services hosted on servers through using a browser client.

Model-View-Controller - The model-view-controller architecture style is present in the systems structural design which is apparent from the existence of the controller module in the middle subsystem. It consists of a number of different controller interfaces corresponding to the different service modules in the back-end subsystem. These different controller interfaces act as mapping tools that supply communication and the transfer of data between different user actions recognized by the GUI in the front-end, to the specific interface in the specific service module for data processing in the back-end. As an example when a user signs in via the GUI, a function in the UserController interface is invoked, where this function makes a call to the specific operation in the SignIn interface in the User Services module to handle the data manipulation and computation related to the sign in action on the back-end..

Layered Style - The system is separated into 3 layers: front-end, middle-end and back-end in addition to containing a DB Access Layer module, located within the back-end subsystem layer which in turn splits the back-end subsystem into a layered component itself. More specifically, it creates a partition between the service modules, the DB Access Layer, and the ecommerce database as a result of the fact that the DB Access Layer is the only component that has access and modification privileges over the ecommerce database, via its different interfaces. These interfaces map the different service modules to specific tables in the ecommerce database. As an example, when a user is signing up or is signed in, which are handled by the SignIn and SignUp interfaces in the user services modules, the user table in the database is accessed via the operations of the UserInfo interface in the DB Access Layer. The

controllerModule also serves as a separate layer routing requests and enabling data transfer between the front and back end.

Pub-Sub - A Pub-Sub architectural style will be used to implement an auction ended page as well as to a payment page that will update as soon as an auction ends. It will also be used to update the highest bidder on an item's bid page when a higher bid is received. This is necessary to give users an equal amount of time to enter a new bid and participate in the auction.

Microservices - The Microservices architectural style is presented in the system's design, where in the back-end subsystem there are several service modules contained. Each service module corresponds to a different service that the system provides, and each contains a set of different interfaces that they provide or are exposed to which allow for these services to be executed. The User service microservice maps to the User Services module; responsible for sign-in and sign up interface operations, the Catalogue service microservice maps to the Catalogue Services module; responsible for searching and selling interface operations, the Payment service microservice maps to the Payment Service module; responsible for pay and receipt interface operations, and lastly the Auction service microservice maps to the Auction Service module; responsible for bid and end interface operations.

Architectural Styles Considered but not chosen:

Monolithic Software Oriented Architecture - Originally, we were considering implementing this kind of architectural style considering that we had originally proposed a single or monolithic component that collectively contained all four services. We decided to instead implement the microservices architecture style in opposition to the monolithic style as a result of the fact that the monolithic style provides higher coupling in the sense that each service is not individually executed at the request of the user, they are not independent of one another or at least not independent of the single component that would have originally contained them, and in the case that if either of the services fail that possibly could affect the others ability to execute considering they belong to the same component; essentially failure of one has the potential to affect the other. As well, this style of design does not allow for each of the services to manage the data individually, where the single monolithic component that contains them does so; a request for data retrieval, access, or modification must initially pass through the monolithic component as opposed to initially through the UserService itself as we have in our system.

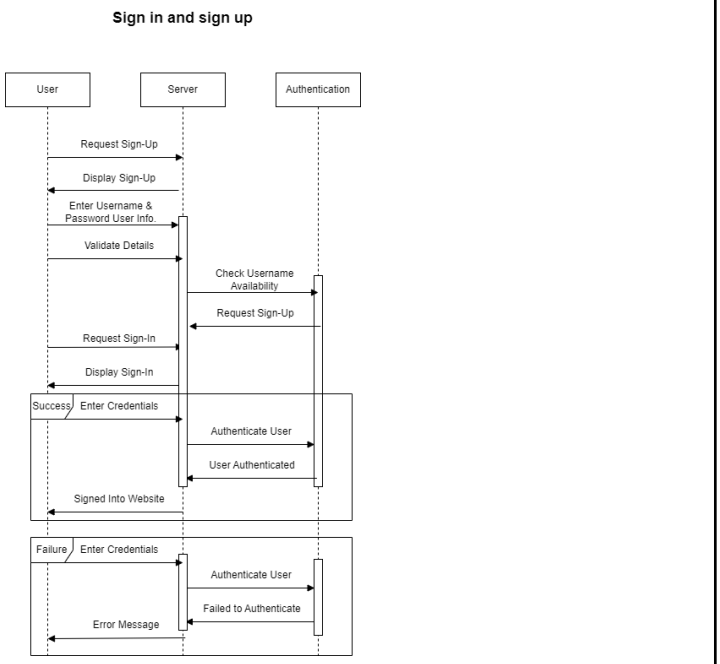
Modularization Criteria:

High Cohesion - The system displays a high level of cohesion as this is demonstrated from the fact that different modules throughout the system contain elements, in this case interfaces, which are highly related to the module that contains them. More specifically, we can see that for each of the service modules in the back-end subsystem, each contains only interfaces that provide operations which allow those services to be provided. Additionally, this can be seen in the DB Access Layer module, where it only contains modules which are single-handedly tasked with providing operations that provide access, retrieval, and modification capabilities to the different tables in the database. Although each interface in the DB Access Layer module is responsible for accessing a single specific table in the database, the fact that they all provide this access functionality is what makes the DB Access Layer a contributor to the system's high cohesion. The same can be said for the Controller module, which only contains controllers; each is related to a unique service, but they are all controllers tasked with providing the transfer and redirect functionality that the Controller module is responsible for.

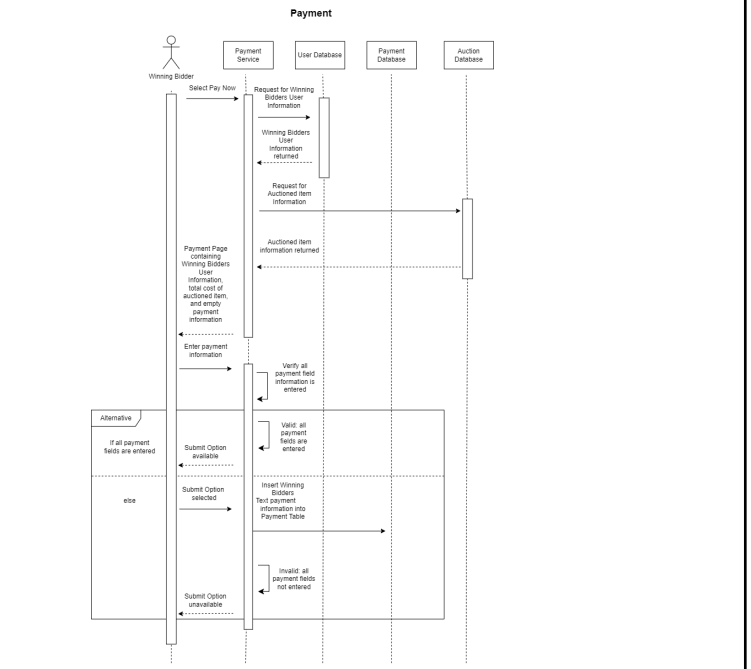
Low coupling & Non-Functional Requirements - The low coupling nature of a microservices architecture ensures high reliability ensuring other services remain operational even if one fails. It also allows for horizontal scaling based on which service is experiencing a high load improving performance without the cost of vertical scaling. Separate services are also great for portability as well as reusability making Microservices optimal for speed, security, reliability, performance, cost and portability of the E-Commerce Website (ECM).

3 Sequence Diagrams

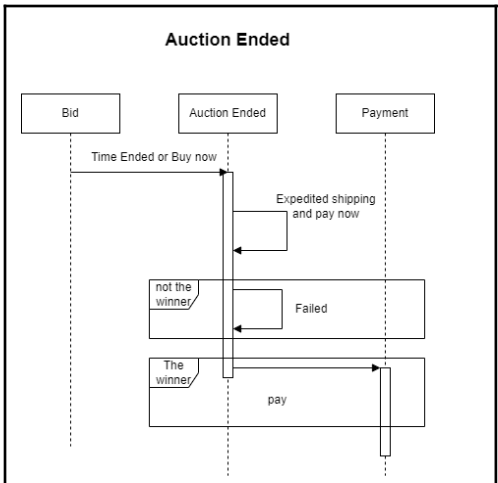
Larger Diagrams are available to view here.



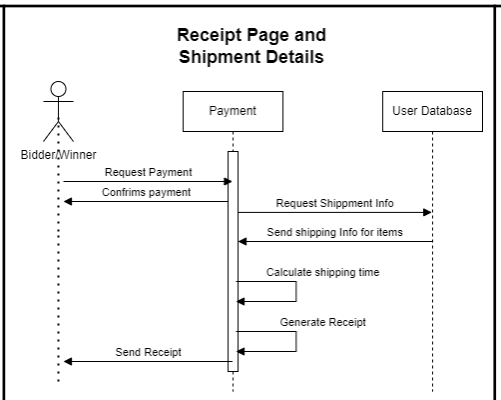
UC1.1 Sign up & UC1.2 Sign in



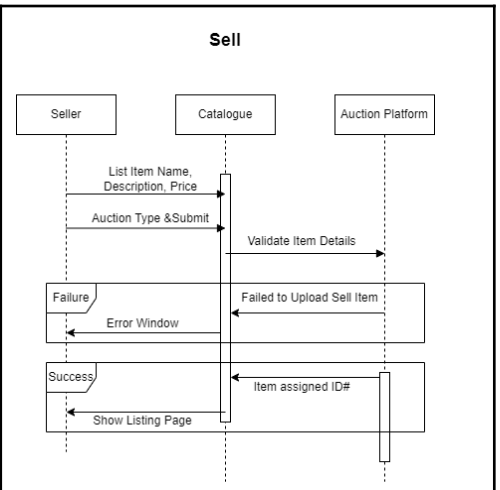
UC5 Payment



UC4 Auction Ended



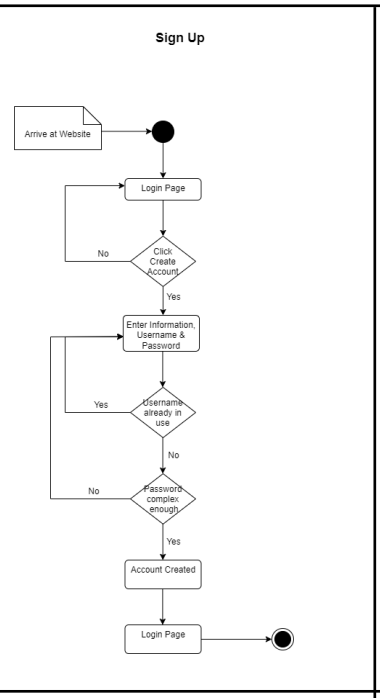
UC6 Receipt Page and Shipment details



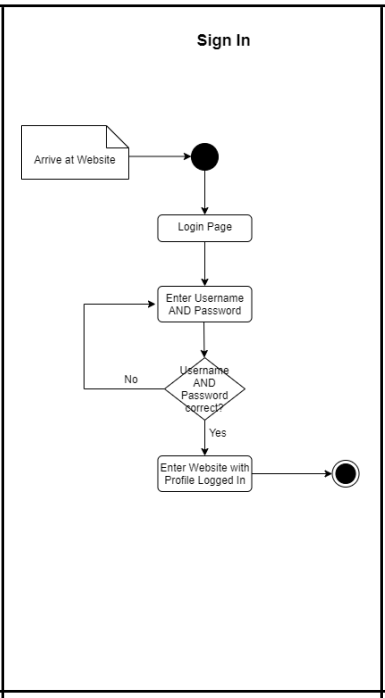
UC7 Sell

4 Activity Diagrams

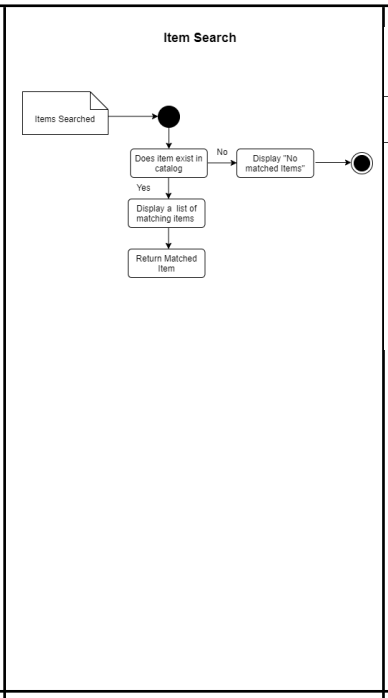
Larger Diagrams are available to view here.



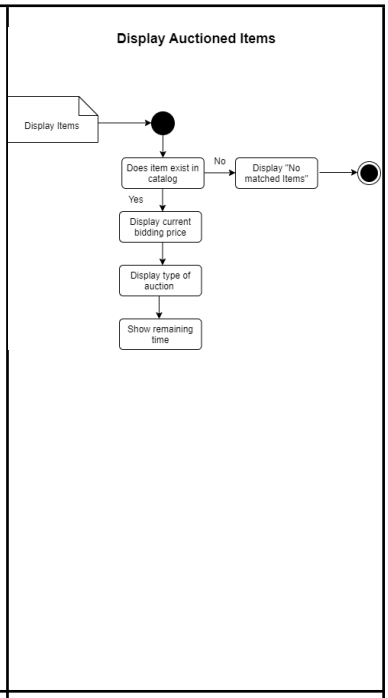
UC1.1 Sign up



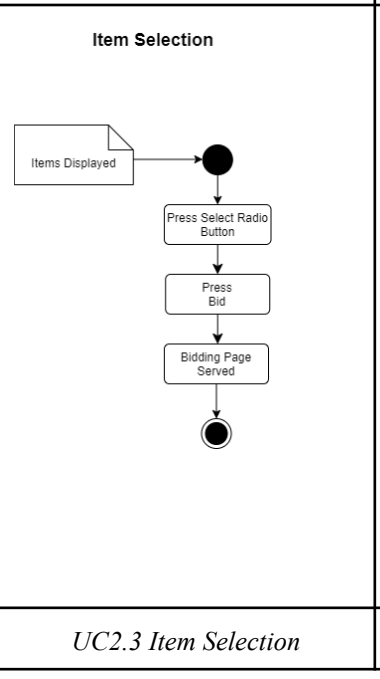
UC1.2 Sign in



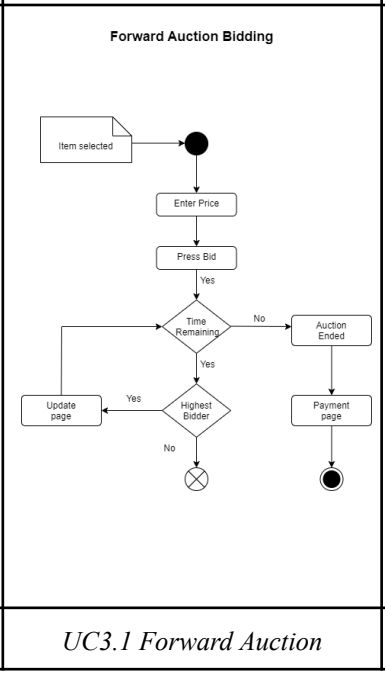
UC2.1 Search



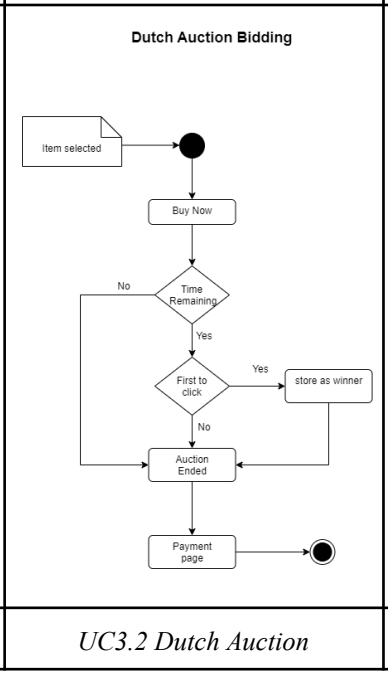
UC2.2 Display Auctioned Items



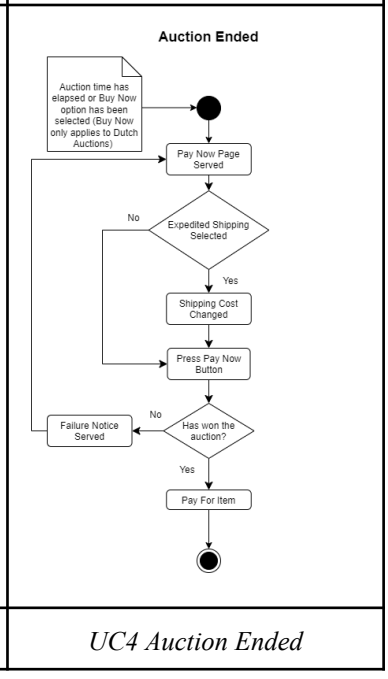
UC2.3 Item Selection



UC3.1 Forward Auction



UC3.2 Dutch Auction

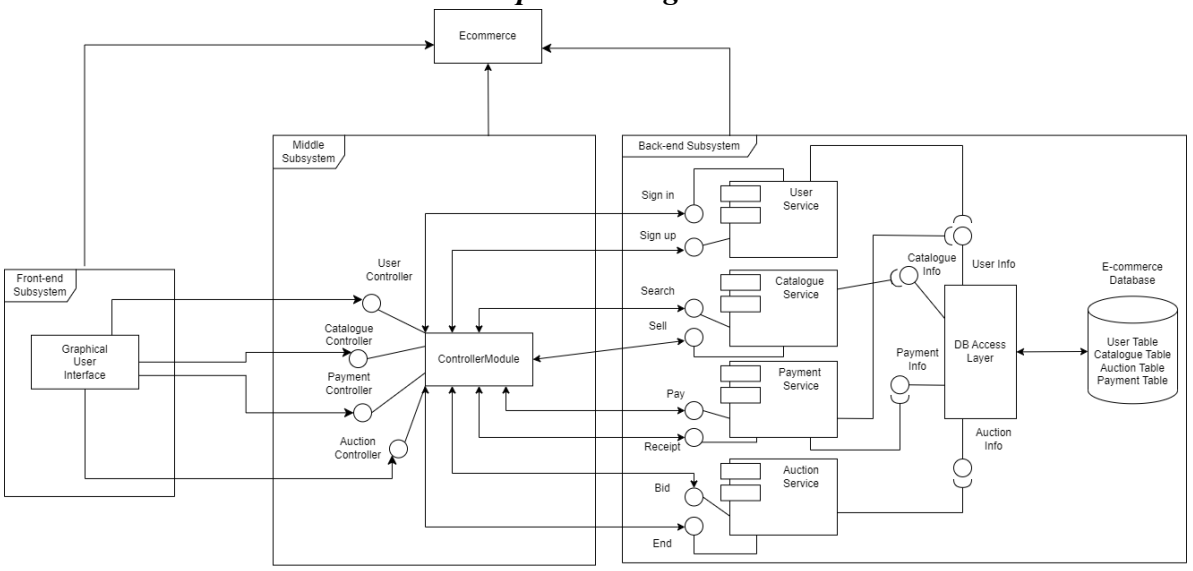


UC4 Auction Ended

5 Architecture

Larger Diagrams are available to view here.

Component Diagram



Modules			
Module Name	**Description**	**Exposed Interface Names**	**Interface Description**
User Services	This module allows users to sign up as a user of the ecommerce system and to sign in.	UserService:SignUp UserService:SignIn	**UserService: SignUp** - allows users to create a user account of the ecommerce system. **UserService:SignIn** - verifies if a user is an existing user of the ecommerce system and permits use of the system for verified existing users.
Auction Services	Is used to access bid information for an item, place a bid on an item and end running auctions based on constraints.	AuctionService:Bid AutionService:End	**AuctionService**:Bid - retrieves an item's bid information, updates highest bidder for forward auction bids and pass bidder to Auction ended for dutch auction bids. **AuctionService: End** - serves an 'auction ended' page for all users. stores winner information
Catalog Services	This module contains interfaces that provide the user with functionality of searching items in the catalogue and placing them for sale on the catalogue as well.	CatologueService: Search CatalogueService: Sell	**CatalogueService: Search** - provides users with the ability to provide the name of the item in a search bar and retrieve a list of items with the same name that are going to be up for auction or are currently under auction. **CatalogueService: Sell** - provides users with the ability to sell items on auctions, whose information they provide.
Payments Services	This module allows the user to make a payment for their winning bid and for them to receive a receipt containing all information regarding their transaction.	PaymentService: Pay PaymentService: Receipt	**PaymentService:** Pay- makes it so that the user can pay for the item that they bid on.The user would first need to have won the bid on an item before being allowed to pay for anything. **PaymentService:** Receipt- User receives a receipt of their order containing their information like their name, shipping address,total amount paid,list of items and the item Ids for each item that they acquired..
ControllerModule	Coordinates control and data flow between other modules, keeps track of auction end time, notifies	ControllerModule: UserController ControllerModule: CatalogController	**ControllerModule: UserControlle**r - maps requests for the sign in and sign up to the UserService module **ControllerModule: CatalogController** - maps requests for the search and sell to the CatalogService module

	users of auction ending, and	ControllerModule: AuctionController ControllerModule: PaymentController	**ControllerModule: AuctionController** - maps requests for the Bid and pay to the AuctionService module **ControllerModule: PaymentController** - maps requests for payment to the PaymentService module
DB access	connects to the database and provides Interfaces for data retrieval and manipulation.	DB access: UserInfo DB access: CatalogInfo DB access: AuctionInfo DB access: PaymentInfo	**DB access: AuctionInfo** - stores, retrieves and updates data from Auction Table: i.e. bids for all items currently being auctioned, the highest price, the highest bidder, and the end time of the auction.

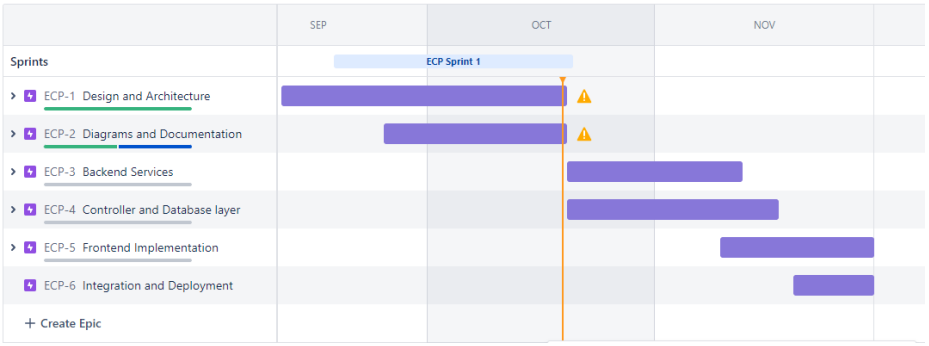
Interfaces		
Interface Name	**Operations**	**Operation Descriptions**
UserService:SignUp	<void> SignUp:POST(String username, String password, String Shipping Address) used by UserService	**UserService:POST(Username, Password, Shipping Information)** store the inputted username, password, and shipping address of the newly signed up user
UserService:SignIn	<Boolean>SignIn:POST(String Username, String Password) used by UserService <String>SignIn:Username:GET(String User) used by DB access <String>SignIn:Password:GET(String Pass) used by DB access	**UserService:GET(Username, Password)** obtain the stored username and password in the authentication server and compare to inputted username and password by user **UserService:POST(AuthenticationResult)** true or false result of whether inputted username and password are correct
CatalogueServices:Search	<List>Search:GET(String name) used by ControllerModule	**CatalogueServices:Search:GET(String name):** invoked when the user enters a keyword in the search bar and selects search, where a list of items from the catalogue with a matching name is returned.
CatalogueServices:Sell	<void> Sell:POST(String name, String description, String auctionType, int time, double startBidPrice) used by ControllerModule	**CatalogueServices:Sell:POST:(String name):** invoked when the user supplies a request to sell an item, where the parameters passed are used to create an item object that will then appear in the catalogue.
PaymentServices:Receipt	<User>Receipt:POST(User user) used by ControllerModule	**PaymentServices:Receipt:POST(User user)**: invoked when the user has won the bid for an item and has made the payment for it as well.It would get all the necessary information about the user.
PaymentServices:Pay	<void>Pay:PUT(double bidprice)	**PaymentServices:PUT(double bidprice):** invoked when the user has won the bid and wants to make a payment for the item they have selected.
AuctionServices:Bid	<Item> Bid:GET(int itemId) used by ControllerModule <Boolean> Bid:PUT(int itemId,int userId, double price) used by ControllerModule	**AuctionServices:Bid:GET(int itemId):** gets an item's current bid information **AuctionServices:Bid:PUT(int itemId, int userId, double price):** pass the bid price by the user when buy button is clicked for forward auctions and returns the success or failed status
AuctionServices:End	<Boolean> End:GET(int itemId) used by AuctionServices and ControllerModule	**AuctionServices:End:GET(int itemId):** records winner to Auction table, ends the auction
DB access:UserInfo	<User>UserInfo: GET(int userId) used by UserServices and PaymentServices <Boolean>UserInfo: POST(User user) used by UserServices <Boolean>UserInfo: DELETE(int userId) used by UserServices	**DB access:UserInfo:GET(int userId):** retrieves the user with the matching userId from the User table in the database, and returns the information in the form of a User object. **DB access:UserInfo:POST(User user):** places the user object passed as the input into the User table to be stored, and a boolean value is returned to confirm success or failure.

		DB access:UserInfo:DELETE(int userId): removes the user from the User table in the database whose id matches the input userId, and a boolean value is returned to confirm success or failure.
DB access: CatalogueInfo	<Item>CatalogueInfo:GET(int itemId) used by CatalogueServices <Boolean>CatalogueInfo:POST(Item item) used by CatalogueServices <Boolean>CatalogueInfo:DELETE(int itemId) used by CatalogueServices	**DB access:CatalogueInfo:GET(int itemId):** searches for the information of the item in the catalogue that has its itemId matched with the input itemId parameter, where once retrieved this information is stored in an Item object to be returned. **DB access: CatalogueInfo:POST(Item item):** places the Item object passed as the input into the Catalogue table to be stored, and a boolean value is returned to confirm success or failure. **DB access:CatalogueInfo:DELETE(int itemId):** removes the catalogue item in the Catalogue table in the database, whose id matches the input itemId, and returns a boolean value to confirm success or failure.
DB access: Payment Info	<Payment>PaymentInfo:GET(int payId) used by PaymentServices <Boolean>PaymentInfo:POST((Payment payment) used by PaymentServices <Boolean>PaymentInfo:DELETE(int paymentId) used by PaymentServices	**DB access:PaymentInfo:GET(int payId):** retrieves the payment from the Payment table in the database whose payId matches the input paymentId. **DB access:PaymentInfo:POST(Payment payment):** places the payment object passed as the input into the Payment table to be stored, and a boolean value is returned to confirm success or failure. **DB access:CatalogueInfo:DELETE(int paymentId):** removes the payment from the Payment table in the database, whose id matches the input paymentId, and returns a boolean value to confirm success or failure.
DB access: Auction Info	<Item>AuctionInfo:GET(int itemId) used by AuctionServices <Boolean>AuctionInfo:POST(Item item) used by CatalogServices <Boolean>AuctionInfo:PUT(int itemId, int price, int user_id) used by AuctionServices	**AuctionInfo:GET(int itemId):** retrieves an item with the given itemId from the Auction table **AuctionInfo:POST(Item):** stores a new item in the Auction table and returns status **AuctionInfo:PUT(int itemId, double price, int user_id)**:updates highest bidder and/or highest price for the item with the given itemId and returns status
ControllerModule : UserController	<void>UserController:GET(String userServiceType, userName, String password) used by GUI	**<void>UserController:GET(String userServiceType, userName, String password):** invoked when the user attempts to execute a user service, as recognized by the GUI, where all of the user's information is passed as inputs to be used to call operations in the sign-up of sign-in interfaces in the UserServices backend module.
ControllerModule : CatalogController	<void>CatalogController:GET(String serviceType, String name, String description, String auctionType, int time, double startBidPrice) used by GUI <void>CatalogController:GET(String serviceType, String name) used by GUI	**<void>CatalogController:GET(String serviceType, String name, String description, String auctionType, int time, double startBidPrice):** invoked when the user attempts to sell an item, as recognized by the GUI, where all of the item catalogue information is passed as inputs to be used to call operations in the sell interface in the CatalogueServices backend module. **<void>CatalogController:GET(String serviceType, String name):** invoked when the user attempts to execute an item search, as recognized by the GUI, where the item's name is passed as an input to be used to call operations in the search interface in the CatalogueServices backend module.
ControllerModule : PaymentController	<void>PaymentController:GET(String serviceType, String userName, double bidprice) used by GUI	**<void>PaymentController:GET(String serviceType, String name):** invoked when the user attempts to execute a payment service, as recognized by the GUI, where the user's username and the bidprice of an item are passed as inputs to be used to call operations in the pay and receipt interfaces in the PaymentServices backend module.
ControllerModule : AuctionController	<void>AuctionController:GET(String serviceType, String userName, String itemId) used by GUI	**<void>AuctionController:GET(String serviceType, String name):** invoked when the user attempts to execute an auction service as recognized by the GUI, where the user's userName and the itemId of the item being bid for in an auction are passed as inputs to be used to call operations in the Bid and End interfaces in the AuctionServices backend module.

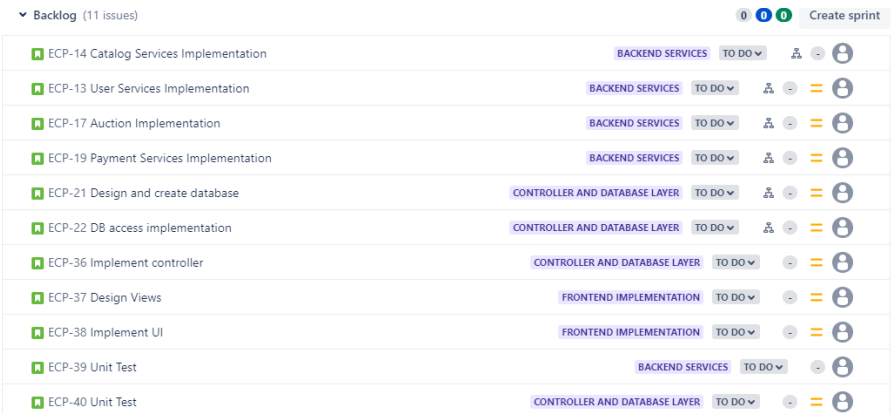
6 Activities Plan

1.1 Gantt Chart, Project Backlog and Sprint Backlog

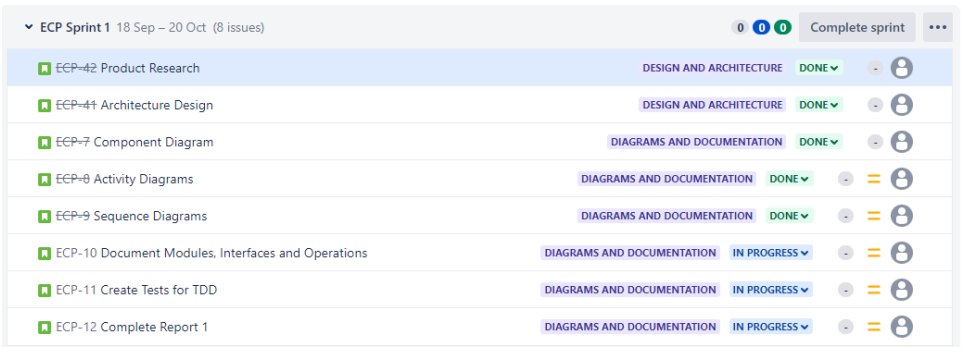
1.1.1 Gantt Chart



1.1.2 Product Backlog



1.1.3 Sprint Backlog



1.2 Group Meeting Logs

Present Group Members	Topic	Decisions made/ Tasks assigned	Meeting Date
Ruth, Kedamawi, Yaqub, Jay	Initial planning	Picked date for next meeting	23/09/2023 4:00 - 4:40 pm
Ruth, Kedamawi, Yaqub, Jay	System Architecture	• UML Diagram Planning • Use-case Activity Diagram Assignments Jay = UC1, Yaqub = UC2.1, UC2.2 Ruth = UC3, Kedi = UC2.3, UC4	24/09/2023 4:00 - 5:10 pm
Ruth, Kedamawi, Yaqub, Jay	System Architecture	• UML Diagram Planning	24/09/2023 10:00 - 11:30 pm
Ruth, Kedamawi, Yaqub, Jay	System Architecture	• UML Diagram Planning, UML Table Assignments Jay: UC6, 7 Yaqub: UC2.1, 2.2, 2.3 Ruth: UC1.1, 1.2, 3 Kedi: UC4, 5 • Use-case Sequence Diagram Assignments Jay: UC7, Yaqub: UC6, Ruth: UC4, Kedi: UC5	06/10/2023 1:00 - 2:00 pm
Ruth, Kedamawi, Yaqub, Jay	System Architecture	• Discussed Design Choices • Designed Modules and Interfaces • Completed Component Diagram	10/10/2023 3:00 - 7:00 pm
Ruth, Kedamawi, Yaqub, Jay	System Architecture & Report	Description of modules, Interfaces and operators & Test cases • Jay = User Services & User Info • Yaqub = Payment Services & Payment Info • Ruth = Auction Services & Auction Info • Kedi = Catalog Services & Catalog Info	12/10/2023 6:00 - 7:00 pm
Ruth, Kedamawi, Yaqub, Jay	System Architecture & Report	Designating and completing remaining tasks • Jay = Overview • Yaqub = Purpose • Ruth = Major Design Decisions • Kedi = Major Design Decisions	19/10/2023 8:00 - 9:00 pm
Ruth, Kedamawi, Yaqub, Jay	Final edits and submission	Going over entire documents, final test cases, links to GitHub established and major design decision explained in detail by everyone	20/10/2023 9:00 - 11:00 pm

2 Test Driven Development

The rest of the test cases can be viewed here:

Test ID	TC002
REST Request	GET/Search?name=item1
Category	Evaluates if existing items in catalogue can be retrieved from Catalogue table in Database, after item search is initiated.
Requirements Coverage	*UC2.1-Succesful-BrowseCatalougeofAuctionedItems-ItemSearch*
Initial Condition	User is signed in.
Procedure	*1. The user types in the name of the item they are looking for.* *2. The user selects the search button.*
Expected Outcome	A list of items matching the keyword the user enters into the search bar; “item1”, is displayed to the user.
Notes	The name of the item that the user provides in the search bar must contain letters, may or may not contain numbers, and must not contain any special characters.