OfferUp!

Design Document

Pouya Sameni - 216491623
Shalom Asbell - 215146954
Adit Jain - 216486136
Sana Khademi- 216326613

Version:	4.0
Print Date:	Feb 27, 2023
Release Date:	Feb 27, 2023
Release Start:	Feb 16, 2023
Approval State:	Approved
Approval By:	Pouya Sameni Shalom Asbell Adit Jain Sana Khademi
Prepared By:	Pouya Sameni Shalom Asbell Adit Jain Sana Khademi
Reviewed By:	Pouya Sameni Shalom Asbell Adit Jain Sana Khademi
Path Name:	OfferUp/root
File Name:	Deliverable 1 Report
Document No:	1

Document Change Control

Version	Date	Authors	Summary of Changes
1.0	2/6/2023	Shalom Asbell Pouya Sameni	1. Document formatted
1.1	2/10/2022	Sana Khademi Pouya Sameni	1. Completed some sections of the design document 2. Imputed some of the diagrams in the document (sequence diagram - UC4 and UC6)
1.2	2/25/2023	Shalom Asbell Adit Jain Pouya Sameni Sana Khademi	1. Added large portions of interface write up 2. Added to the Test Driven Development section. 3. Worked with Adit to Review Sequence Diagrams 4. Added the Tasks to be completed for future sprints
1.3	2/26/2023	Shalom Asbell Pouya Sameni Adit Jain	1. Completed and formatted UC1, UC2, UC3, UC5, UC6 activity diagrams. 2. Completed Sequence Diagrams 3. Completed Major Design Decisions write up. 4. Completed Services Interfaces Methods
1.4	2/26/2023	Sana Khademi Pouya Sameni	1. Completed UC4, UC6 activity diagram

Document Sign-Off

Name (Position)	Signature	Date
Pouya Sameni		2/27/2023
Shalom Asbell		2/27/2023
Adit Jain		2/27/2023
Sana Khademi		2/27/2023

Table of Contents

1 Introduction

1.1 Purpose

The goal of this project is to create a fully functional ecommerce auctioning web application similar to the popular e-commerce platform eBay. Specifically, we will allow users to create accounts, log into our application, search for items under auction, bid on and purchase items in *forward auctions*, and directly purchase items in *dutch auctions*. Specific project requirements are defined by six use cases outlined in the ***Project Description*** document [1].

1.2 Overview

The purpose of this document will be to cover the main design choices for our auction system and how they fulfil both the use case requirements and the modularization criteria specified by the ***Deliverable 1*** description [2]. Sequence diagrams and activity diagrams will be presented for the six use cases: sign-in and sign-up, item selection and search, bidding and auction summary for *dutch* and *forward* auctions, auction ended, payment page, receipt, and shipment information [1]. With these diagrams we will create a visual depiction of the flow of messages between objects and the steps needed to fulfil a use case. The overview architecture of our system, including all of its services, databases, and connection points, will be covered in **Section 6**. The design decisions made to achieve the modularization criteria will be covered in **Section 3**. **Section 4** and **Section 5** contain sequence and activity diagrams that describe how the various components of our system interact to achieve the use case requirements. **Section 7** will cover project planning and **Section 8** will cover system testing.

1.3 References

[1] Kontogiannis, K. *Project Description*, https://eclass.yorku.ca/pluginfile.php/4698816/mod_resource/content/1/EECS-4413-Project-Description-Use-Cases-README-FIRST-POSTED.pdf, accessed on 2/6/2023.
[2] Kontogiannis, K. *Deliverable 1*, https://eclass.yorku.ca/pluginfile.php/4698818/mod_resource/content/1/EECS4413-Deliverable-1-Instructions-POSTED.pdf
[3] Hafeez, Usama. "Microservice Architecture, Its Design Patterns and Considerations." *C# Corner*, https://www.c-sharpcorner.com/article/microservice-architecture-its-design-patterns-and-considerations/, accessed on 2/22/2023.
[4] "Facade." *Refactoring.Guru*, https://refactoring.guru/design-patterns/facade.

[5] Yıldız, S. "N-Tier Architecture Explained." *Medium*, *https://medium.com/geekculture/n-tier-architecture-explained-5d2e0246c354, accessed on 2/22/2023.*
[6] O'Riordan, M. "Everything You Need To Know About Publish/Subscribe." *Ably*, https://ably.com/topic/pub-sub, accessed on 2/22/2023.

2 Major Design Decisions

We have opted to employ a five-tier system architecture consisting of the **Client**, **Frontend**, **Middleware**, **Backend**, and **Database** tier to develop our project, in accordance with the principles of multi-tier architecture [5]. Because the tiers are logically separate, we give our system the ability to scale and distribute each tier individually. Each tier will be described in detail below:

2.1 FrontEnd Tier

The FrontEnd tier is composed of two components: the **UIFacade** component and **UIViews** component. The UIFacade component exposes a REST API that accepts HTTP requests from the client and subsequently calls upon the UIViews component based on the information stripped from the requested URL. UIView's then generates a complete user view by making an appropriate call to the Middleware to retrieve or update data on the Backend, populates a template view with the latest data, and returns the complete view to UIFacade who returns the view to the client. By adding the additional layer of indirection between UIFacade and UIViews, we decouple client HTTP requests from calls to the Middleware and view construction. From the browsers perspective, they simply interact with our system through the REST API exposed by UIFacade and are returned an appropriate view. In this manner, the UIFacade implements a *facade pattern* [5].

The UIFacade component also implements a modified version of the vanilla *publisher-subscriber* pattern [6] to allow for persistent updates of auction pages. For example, when entering a *forward auction*, a client will want to receive persistent updates on the current highest bid for the item up for auction. In this scenario, the client ForwardAuctionView will create a socket connection with the UIFacade and write into the socket the itemID of the item that they are subscribing to receive bid updates for. Any subscribed client can also write into the socket a number, itemID, and their userID in order to bid on an item; that is, subscriber clients can also act as publishers. When the UIFacade receives this information, it will attempt to perform the bid on the item by a call to UIView which will return a view string containing the latest highest bid on the item. UIFacade then writes the view string into sockets subscribed to the itemID.

2.2 Backend Tier

In any e-commerce system, it is crucial to divide the backend into smaller, modular components, each with their own distinct duty. This allows for improved maintainability and scalability of the system by allowing each component to be tested and maintained independently. In following this principle, we have chosen a *microservice architecture* [3] to implement our back-end. We have divided our back-end into multiple modularized services that implement a small subset of Backend functionality: **CatalogueService**, **AuctionService**, **PaymentService**, and **AccountService**. These services never communicate with each other and each has their own independent, single-table databases: **CatalogueDB**, **AuctionDB**, **PaymentDB**, and **AccountDB**. The result of these architectural choices is complete independence of service behaviour, meaning our services are loosely coupled. Additionally, our service components are highly cohesive because their operations are related by the single-table database they access (e.g., PaymentDB) and by extension, have a semantically well defined role (e.g., handling payments).

2.3 Middleware Tier

To fulfil complex Backend requests that need one or more services, in place of communication between services, we make use of a *facade* component called the **Controller** that delegates parts of the request to different services. Based on the responses received from the services, the Controller generates a response that is then returned to the client. To the client, the inner workings of the Controller class are completely hidden. In this manner, the Controller implements a *facade pattern* [4].

2.4 Database Tier

As described in section **2.3**, our databases are single-tabled and correspond directly to the services that use them, allowing our Backend services to be loosely coupled and highly cohesive. The tables of each database are the following (***** denotes the primary key):

Account DB
*UserName
Email
FirstName
LastName
HashedPassword
StreetAddress
StreetNumber
PostalCode
City
Country

Auction DB
*AuctionID
AuctionType
HighestBidderUserID
HighestBid
InitialPrice
BidderID
ItemID
StartTimeOfAuction
EndTimeOfAuction
DecrementAmount

Catalogue DB
*ItemID
ItemName
Description
ExpeditedShippingFee
ExpeditedShippingTime
RegularShippingFee
RegularShippingTime
ItemSold

Payment DB
*PaymentID
UserID
TotalPrice
PurchaseDate
ItemID

3 Sequence Diagrams

3.1 UC1: Sign-Up and Sign-In

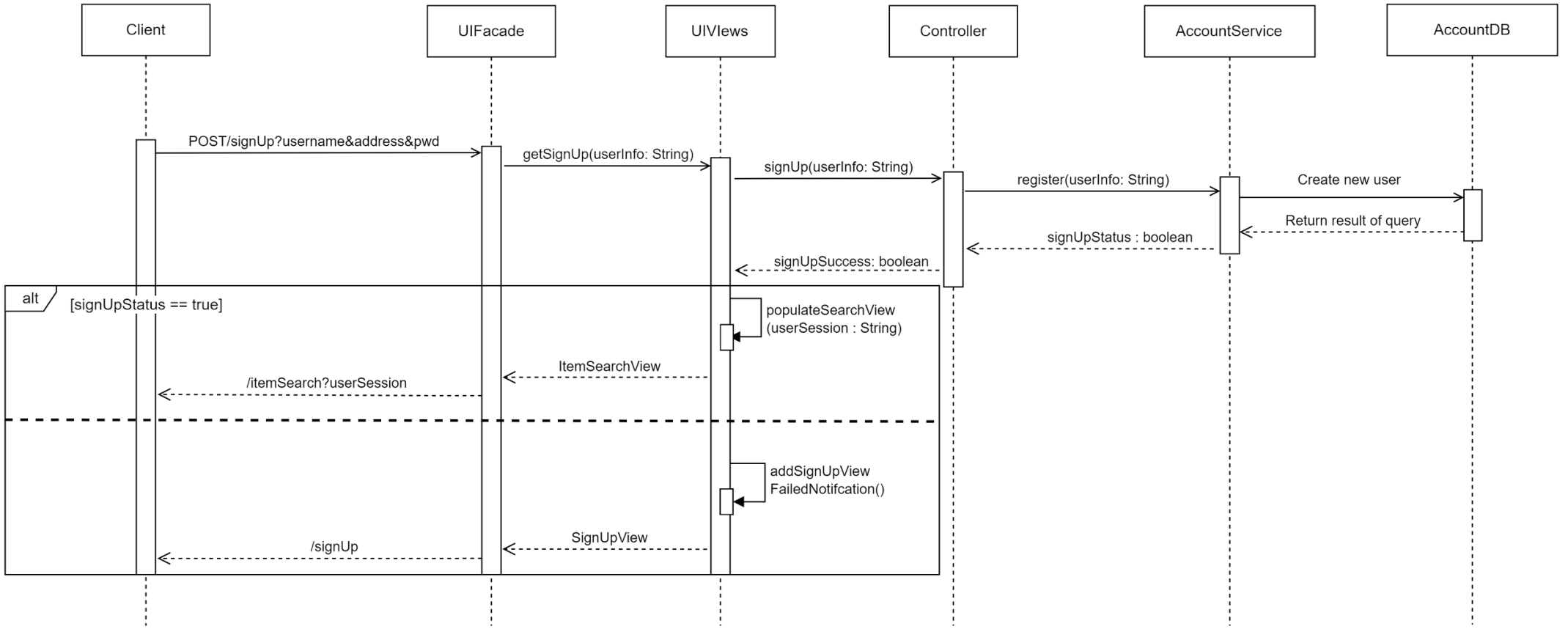


Fig. 1. *UC 1.1 Sign-Up* - user sign-up sequence. Client is assumed to be on the *Sign-Up* user interface at the beginning of the sequence.

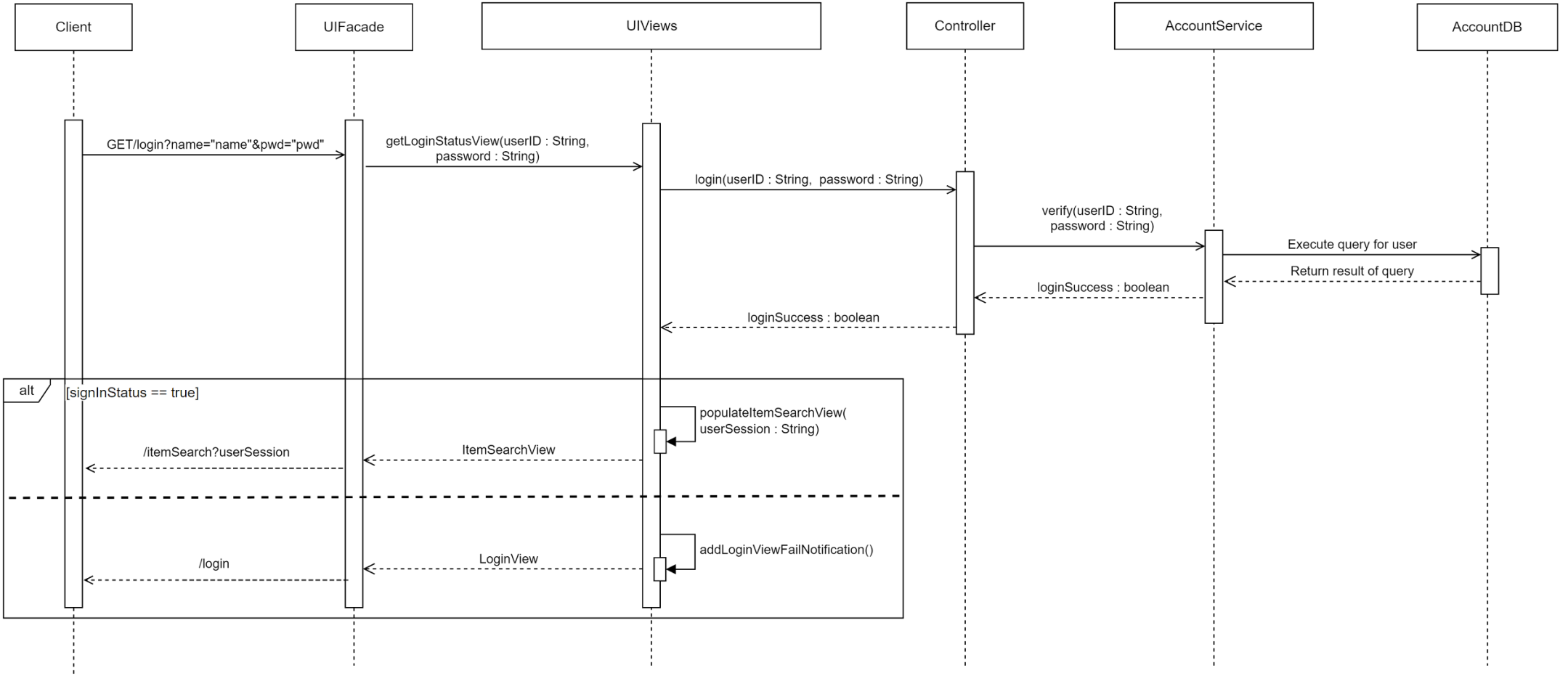


Fig. 2. *UC 1.2 Sign-In* - user sign-in sequence. Client is assumed to be on the *Sign-In* user interface at the beginning of the sequence.

3.2 UC2: Browse Catalogue of Auctioned Items

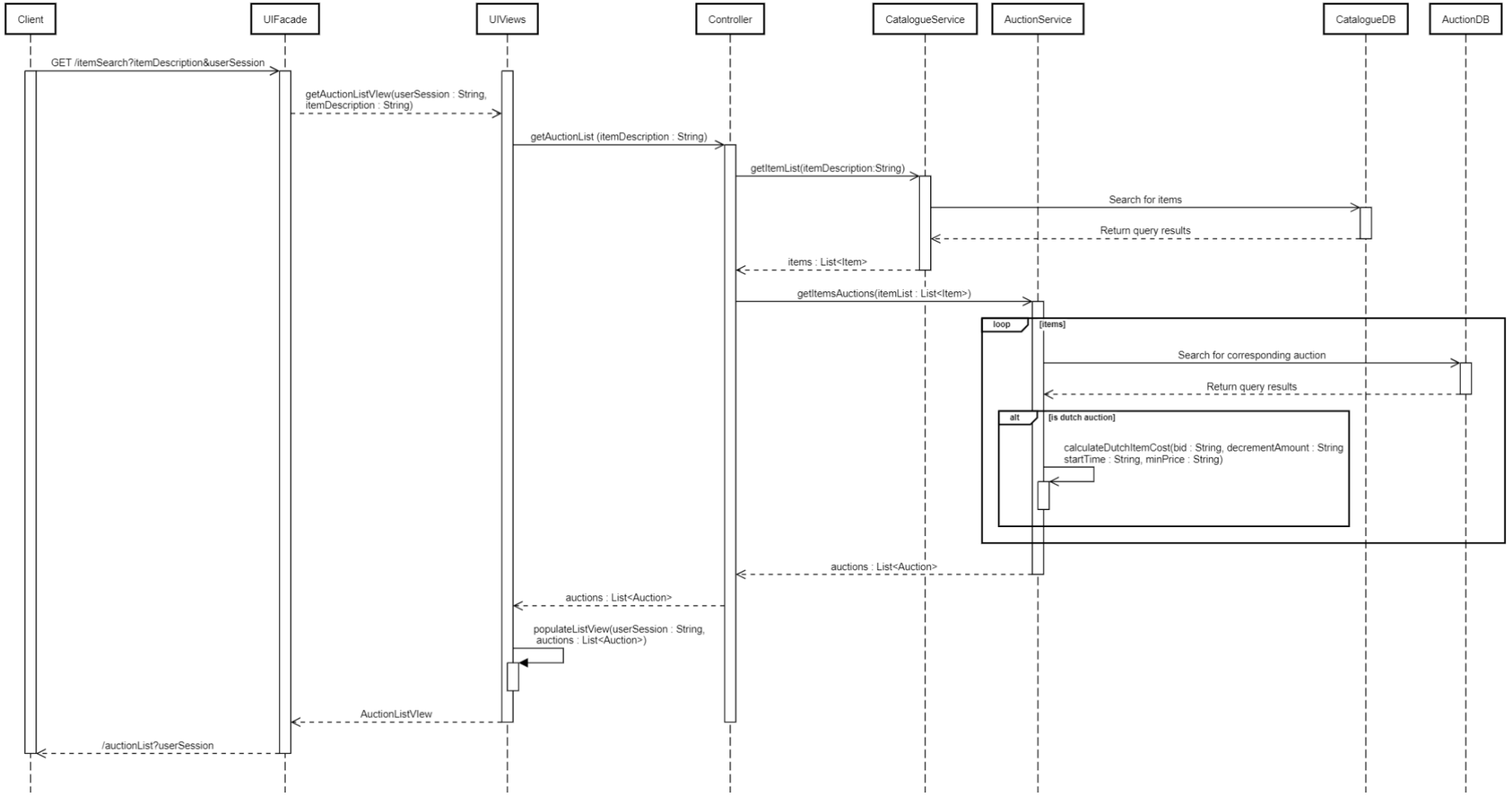


Fig. 3. *UC 2.1 Item Search* - item search sequence. Client is assumed to be on the *Item-Search* user interface at the beginning of the sequence.

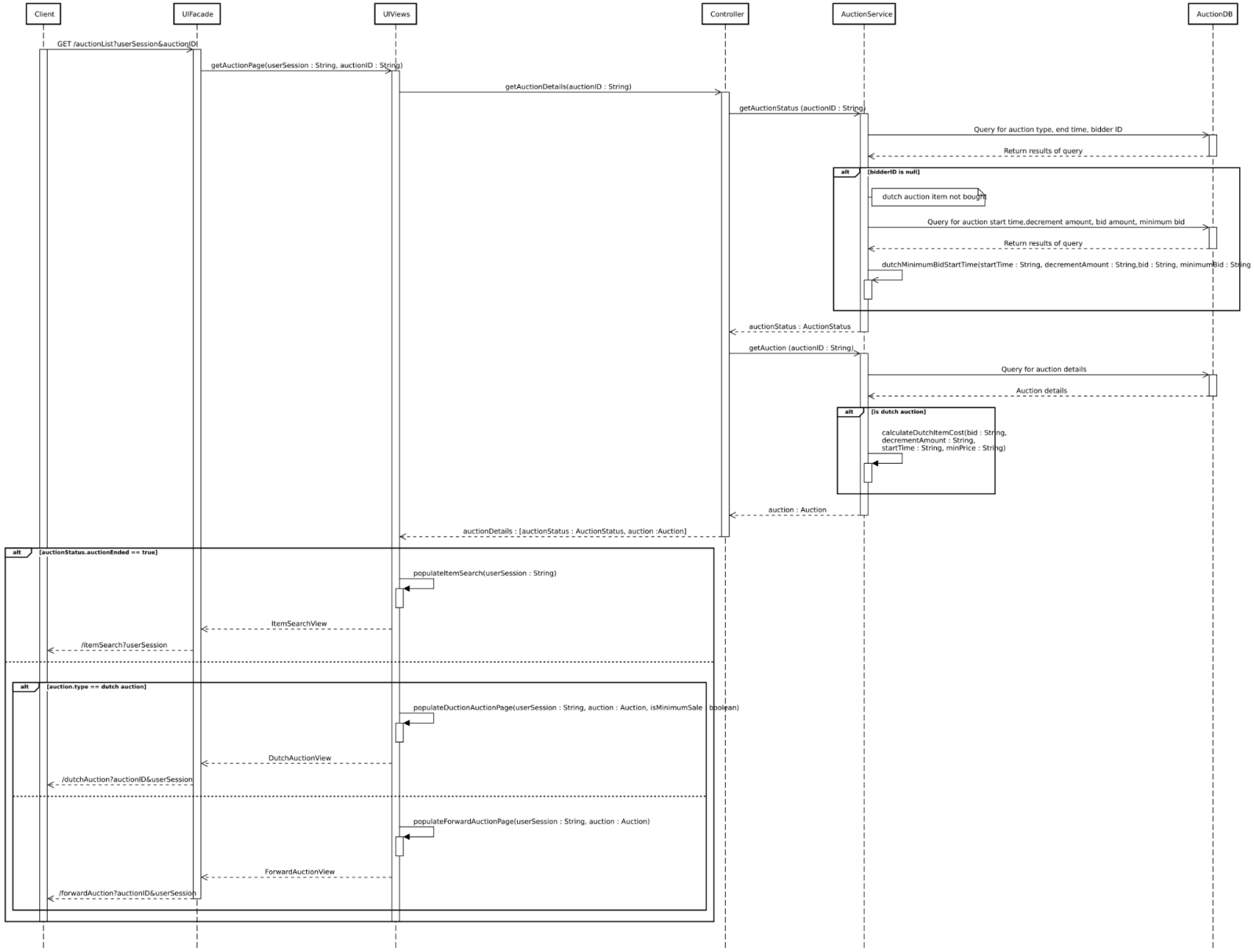


Fig. 4. *UC 2.2 Display Auctioned Items* & *UC 2.3 Item Selection*. Client assumed to be on *Display Auctioned Items UI* at the beginning of the sequence.

3.3 UC3: Bidding

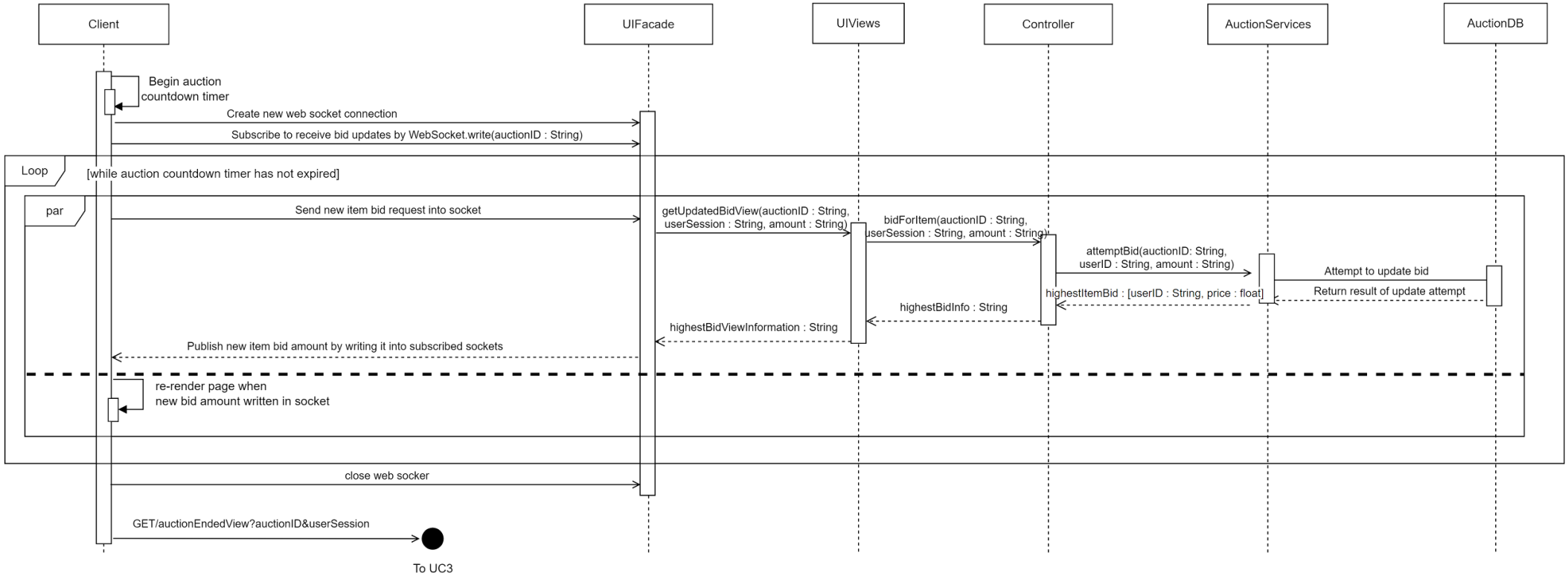


Fig.5. *UC 3.1 Forward Auction Bidding*. Client assumed to be on *Forward Auction Bidding UI* at the beginning of the sequence. See **Fig.7** for completion of sequence (subsequence titled UC3.0 because it is used by UC3.1 and UC3.2).

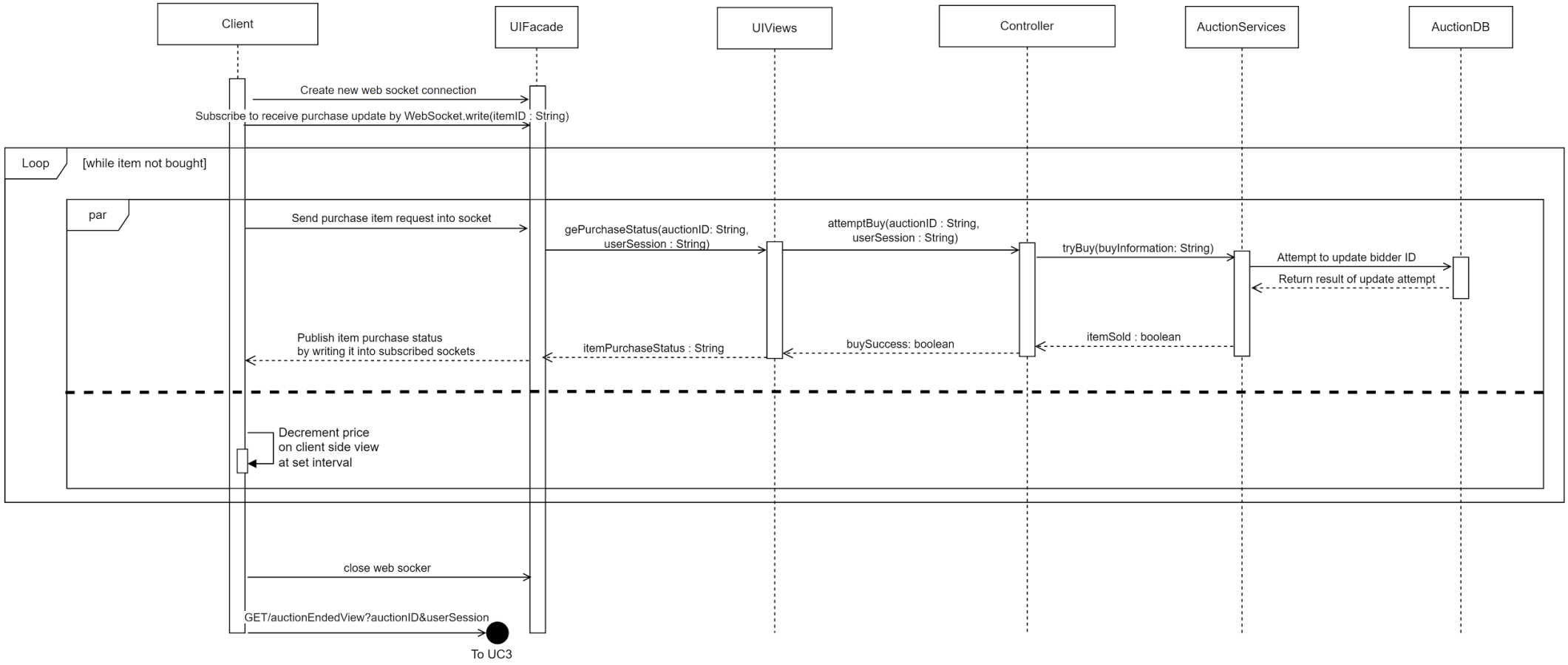


Fig.6. *UC 3.2 Dutch Auction Bidding*. Client assumed to be on *Dutch Auction Bidding UI* at the beginning of the sequence. See **Fig.7** for completion of sequence (subsequence titled UC3.0 because it is used by UC3.1 and UC3.2).

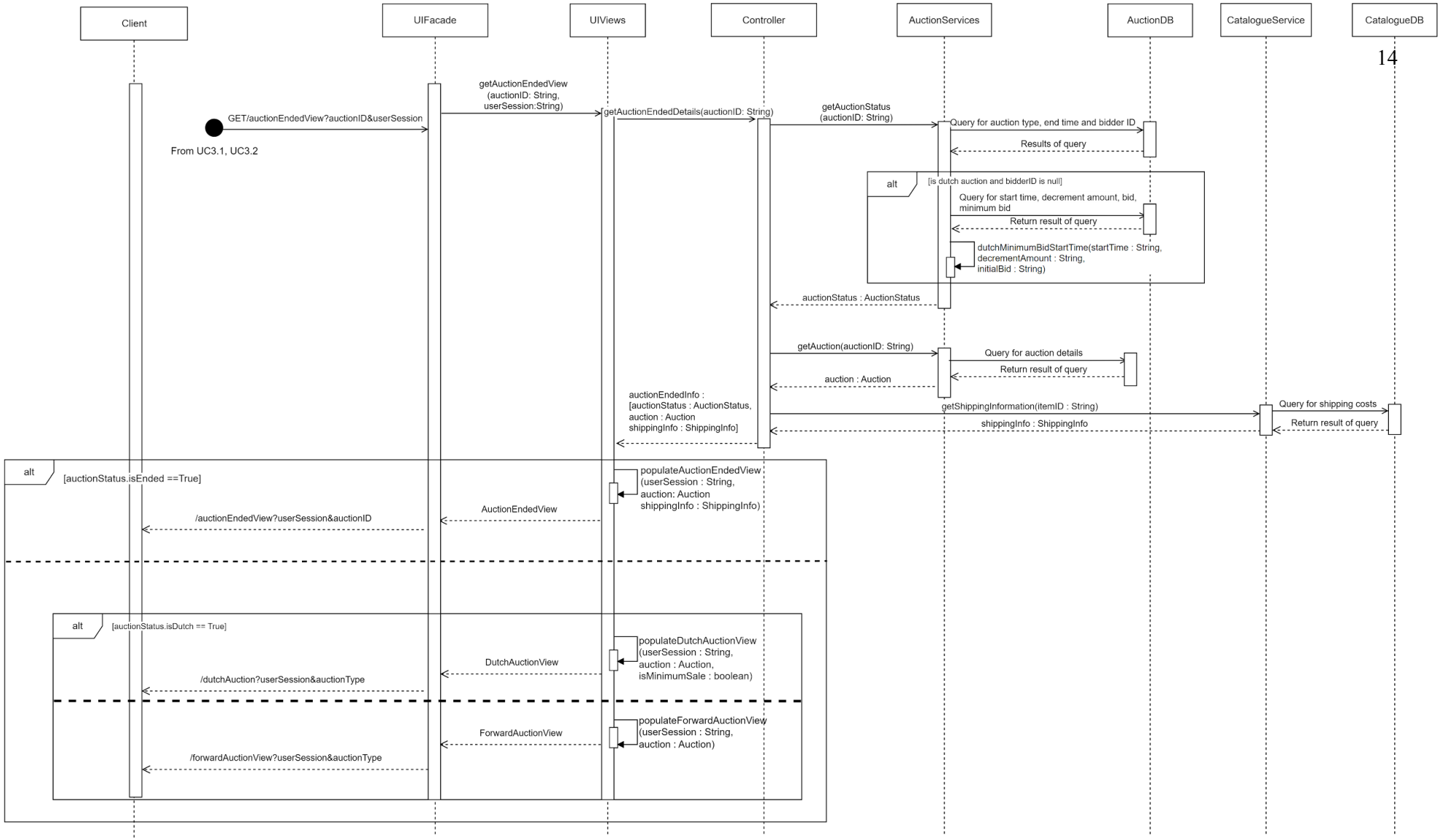


Fig.7. *UC 3.0* Sequence that is used by both *UC 3.1* and *UC 3.2* (see **Fig.5.** and **Fig.6**).

3.4 UC4: Auction Ended

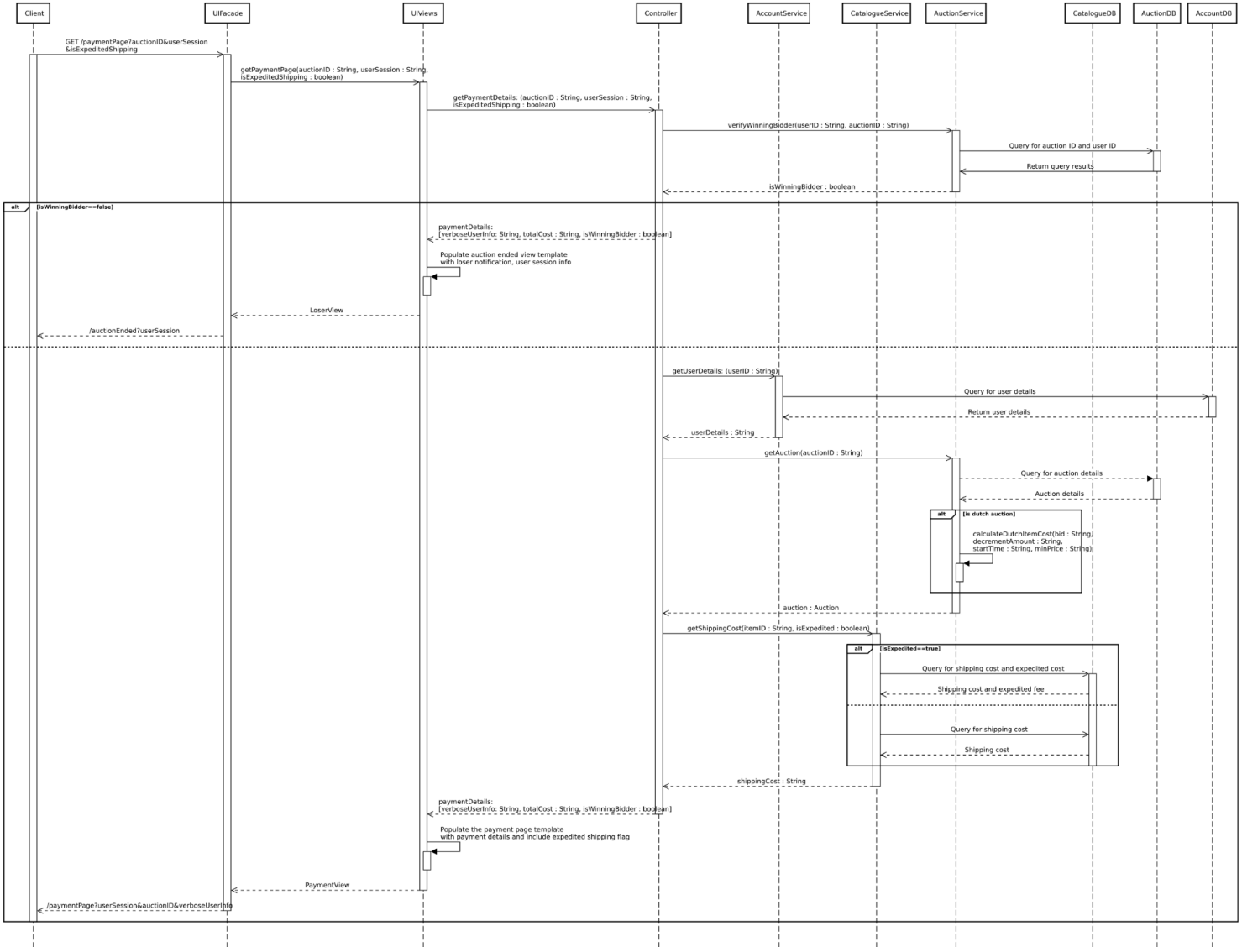


Fig.8. *UC4 Auction Ended* activity. Client assumed to be on *Auction Ended* UI at the start of the activity.

3.5 UC5: Payment

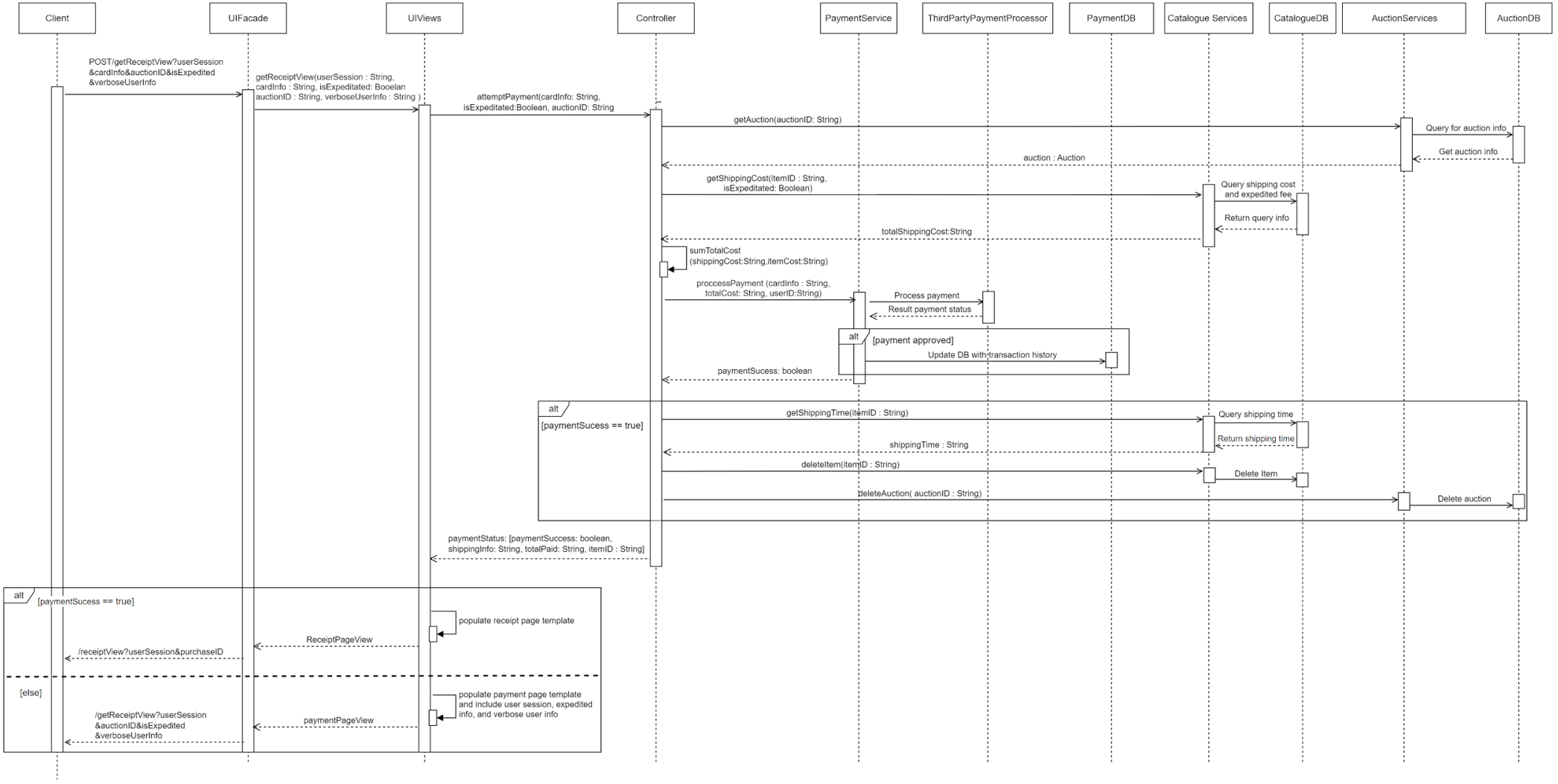


Fig.9. *UC 5.0 Payment* sequence. Client assumed to be at *Payment Page UI* at beginning of sequence.

3.6 UC6: Receipt Page and Shipment Details

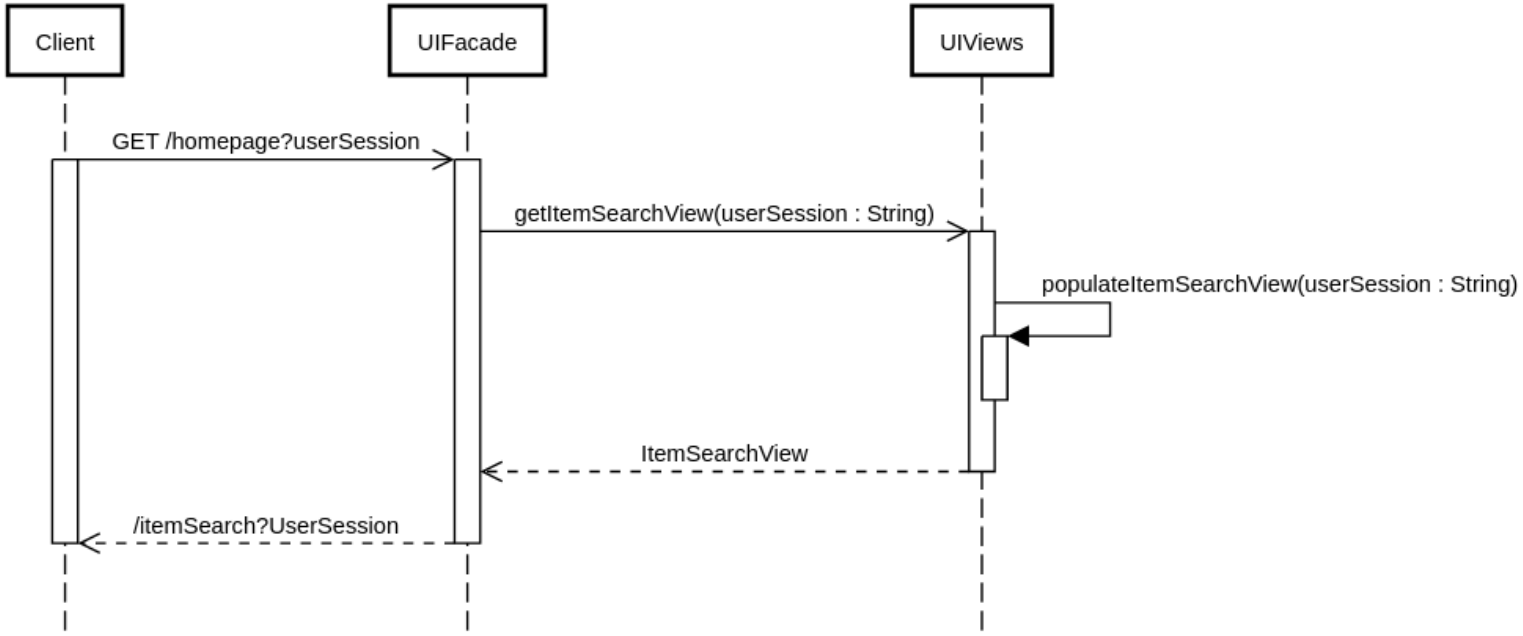


Fig.10. *UC 5.0 Recept Page and Shipment Details* sequence. **Client is assumed to be at the Receipt *and Shopping Details* UI at the beginning of sequence.** Thus, the sequence represents a simple navigation to our homepage which we chose to be the *Item Search View*. See **Fig.9** to confirm that the receipt page and all of its details was returned to the client as appropriate.

5 Activity Diagrams

5.1 UC1: Sign-Up and Sign-In

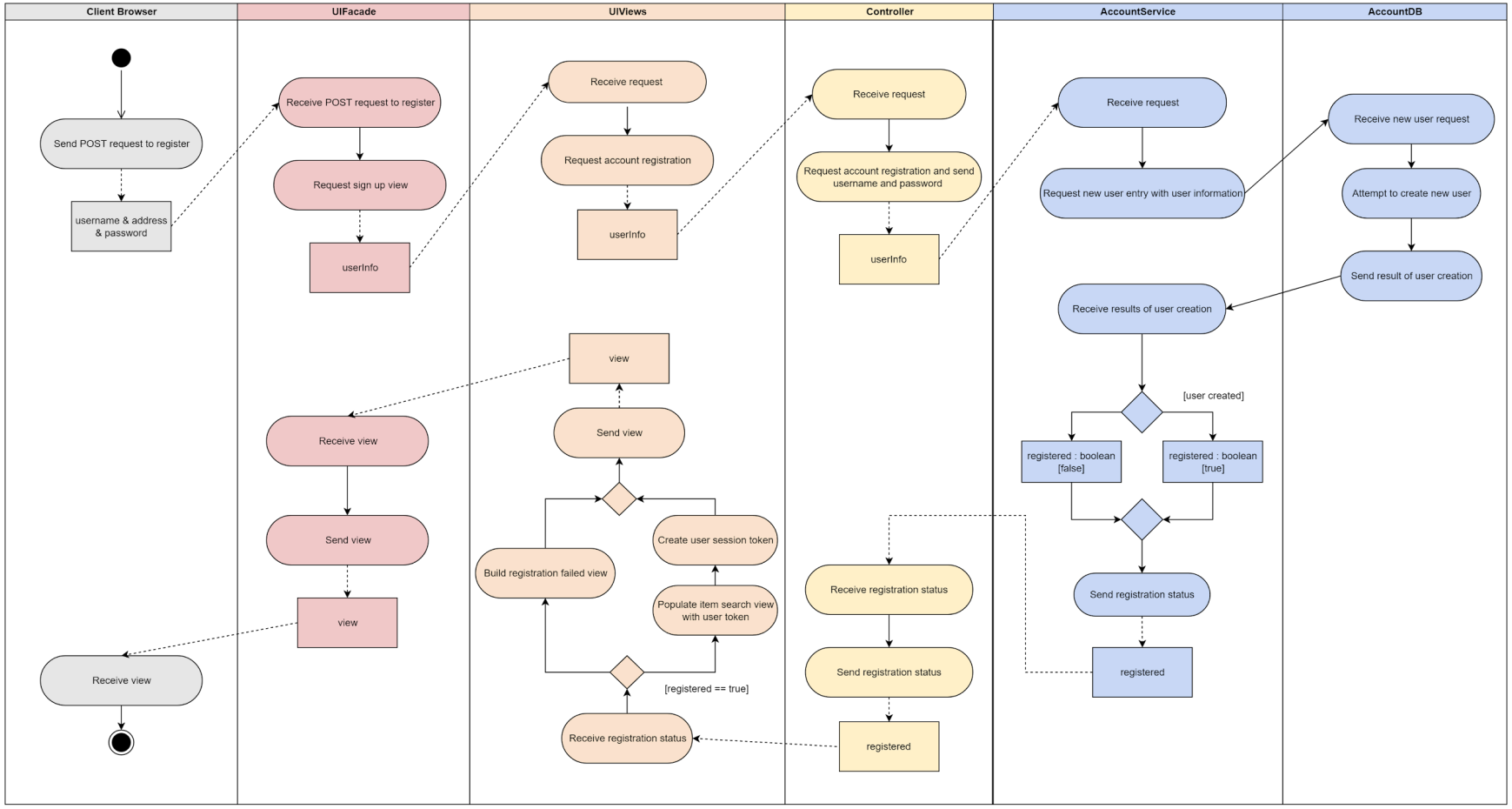


Fig. 11. *UC 1.1 Sign-Up* - user sign-up activity. Client is assumed to be on the *Sign-Up* user interface at the beginning of the activity.

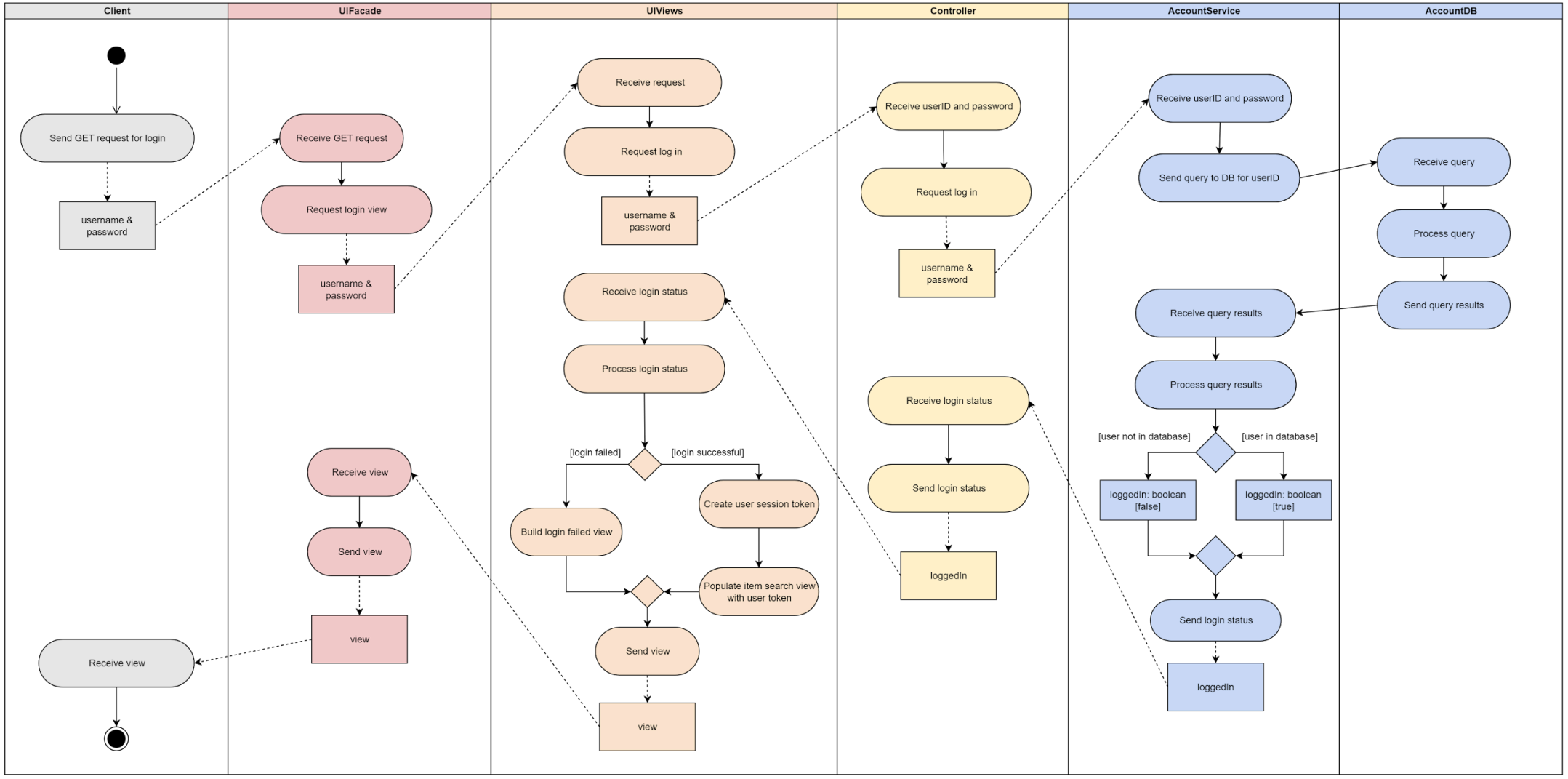


Fig. 12. *UC 1.2 Sign-In* - user sign-in activity. Client is assumed to be on the *Sign-In* user interface at the beginning of the activity.

5.2 UC2: Browse Catalogue of Auctioned Items

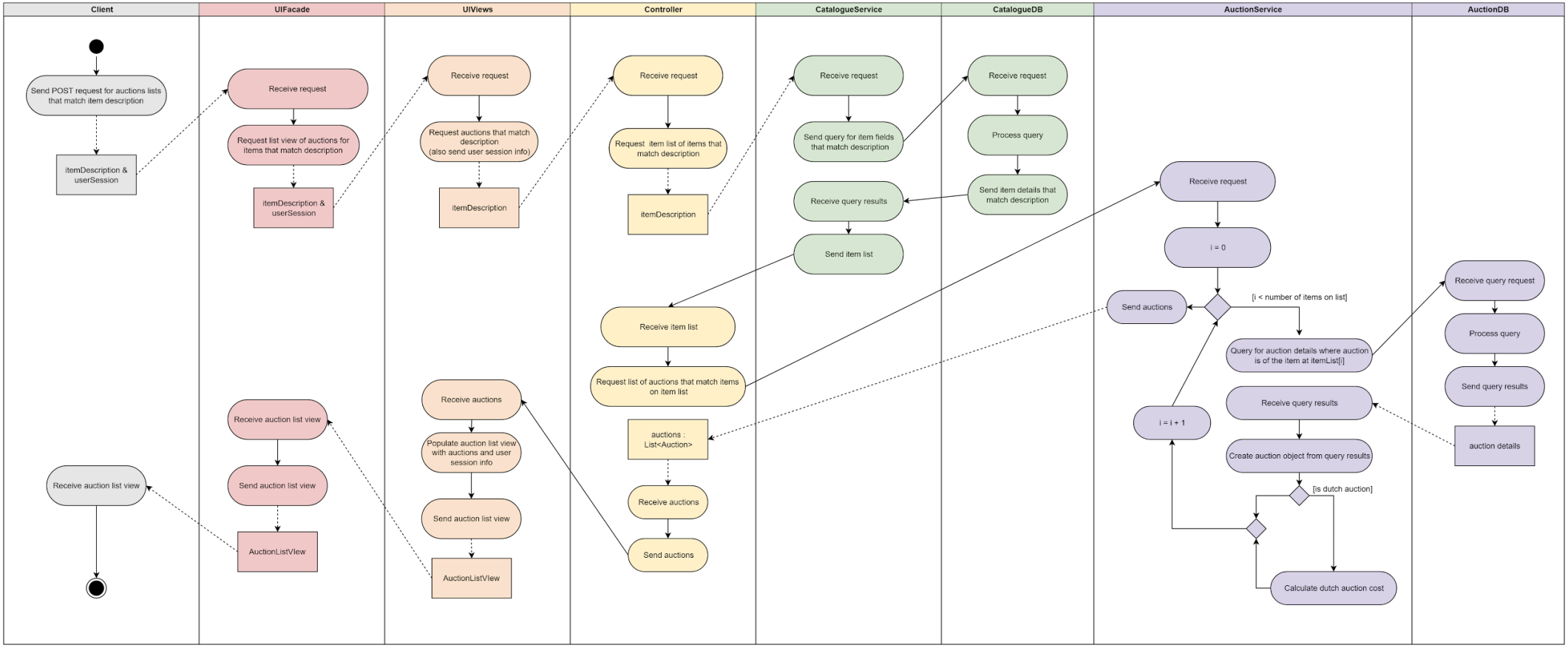


Fig. 13. *UC 2.1 Item Search* - item search activity. Client is assumed to be on the *Item-Search* user interface at the beginning of the activity.

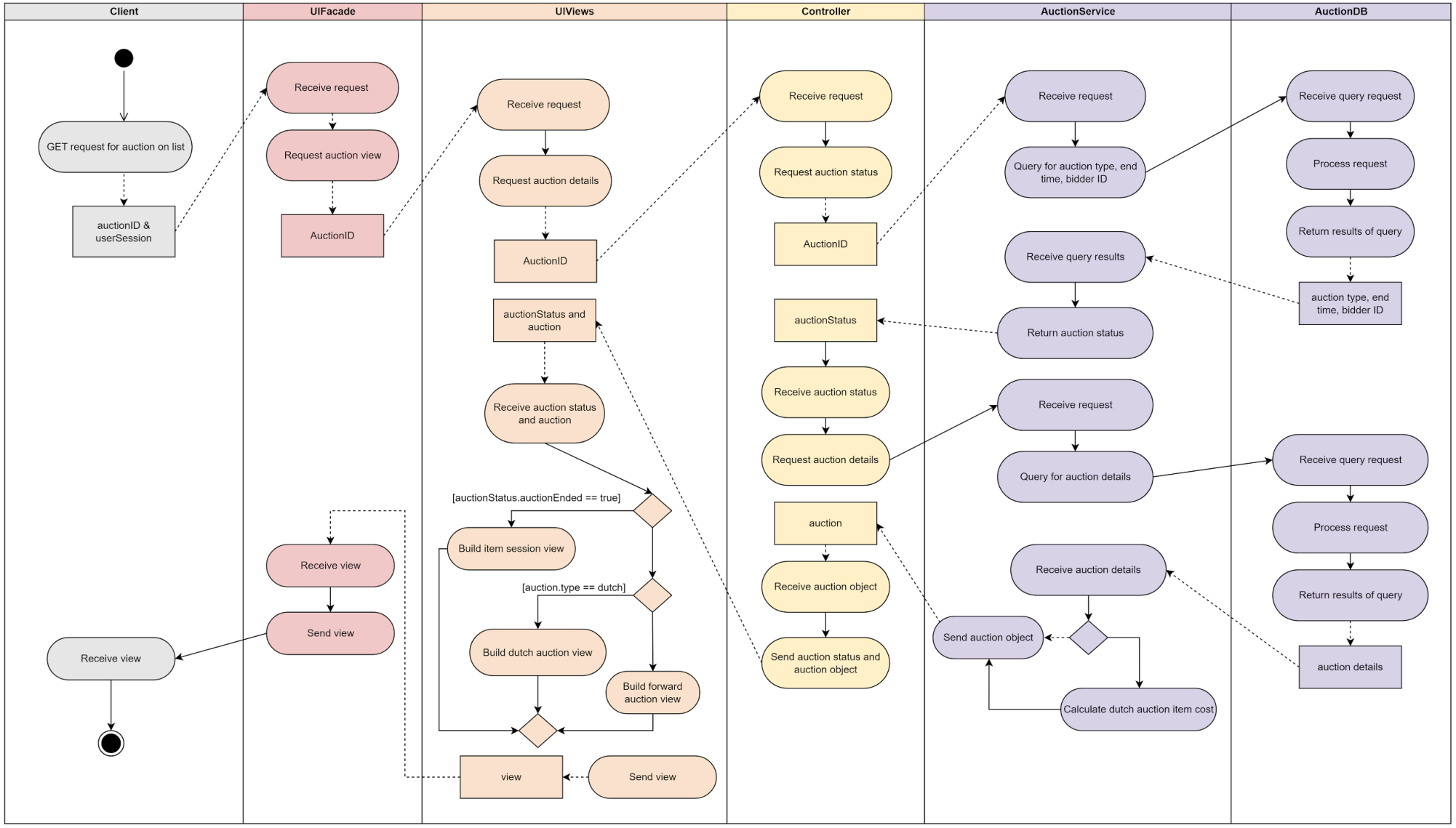


Fig. 14. *UC 2.2 Display Auctioned Items* & *UC 2.3 Item Selection*. Client assumed to be on *Display Auctioned Items UI* at the beginning of the activity.

5.3 UC3: Bidding

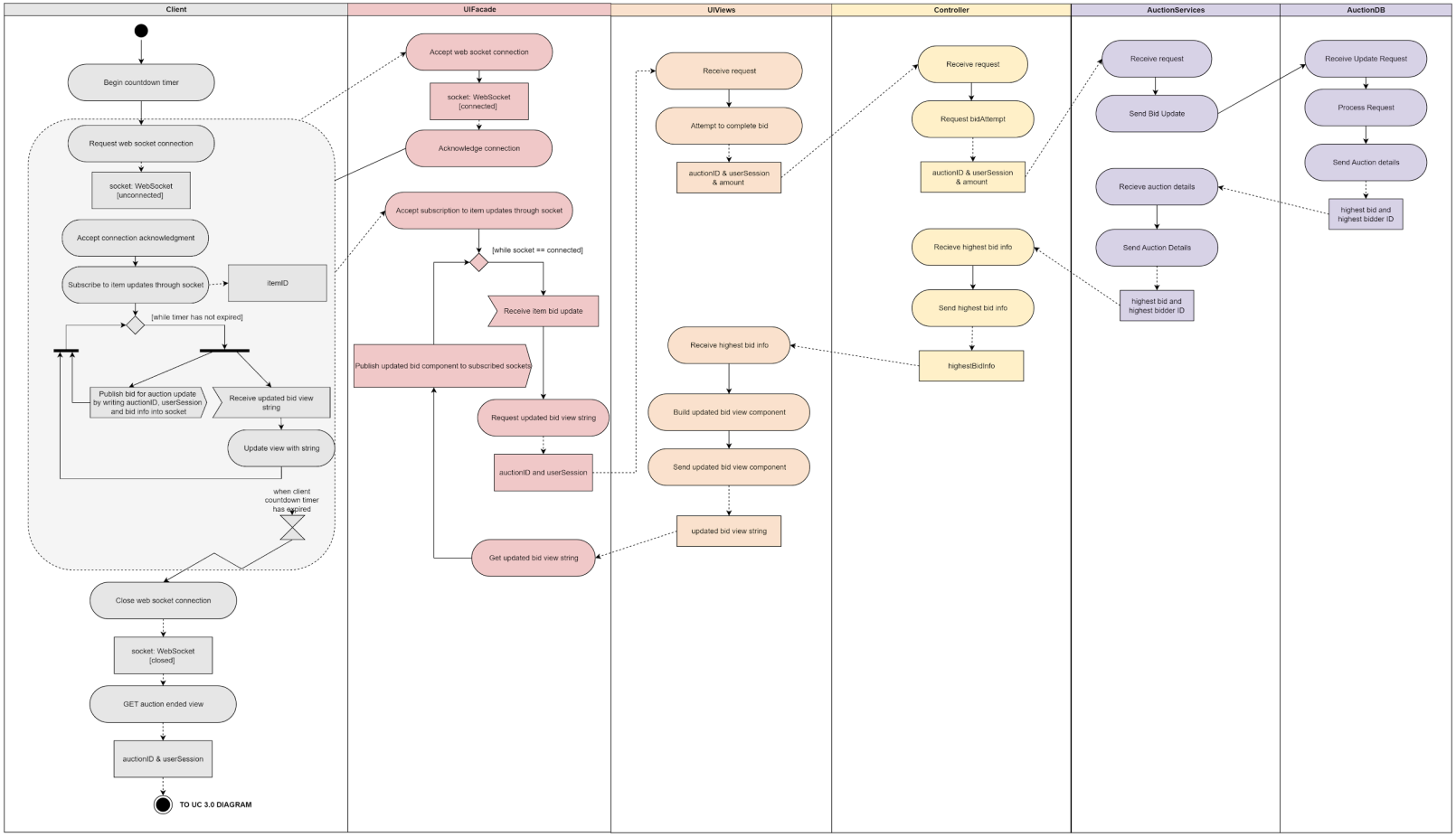


Fig. 15. *UC 3.1 Forward Auction Bidding*. Client assumed to be on *Forward Auction Bidding UI* at the beginning of the activity. Notice that the ending activity UC3.1 is very similar to UC4.0, so refer to UC4.0 for what an activity of UC3.0 would look like. To see the sequence diagram of UC3.0 refer to **Fig.7**.

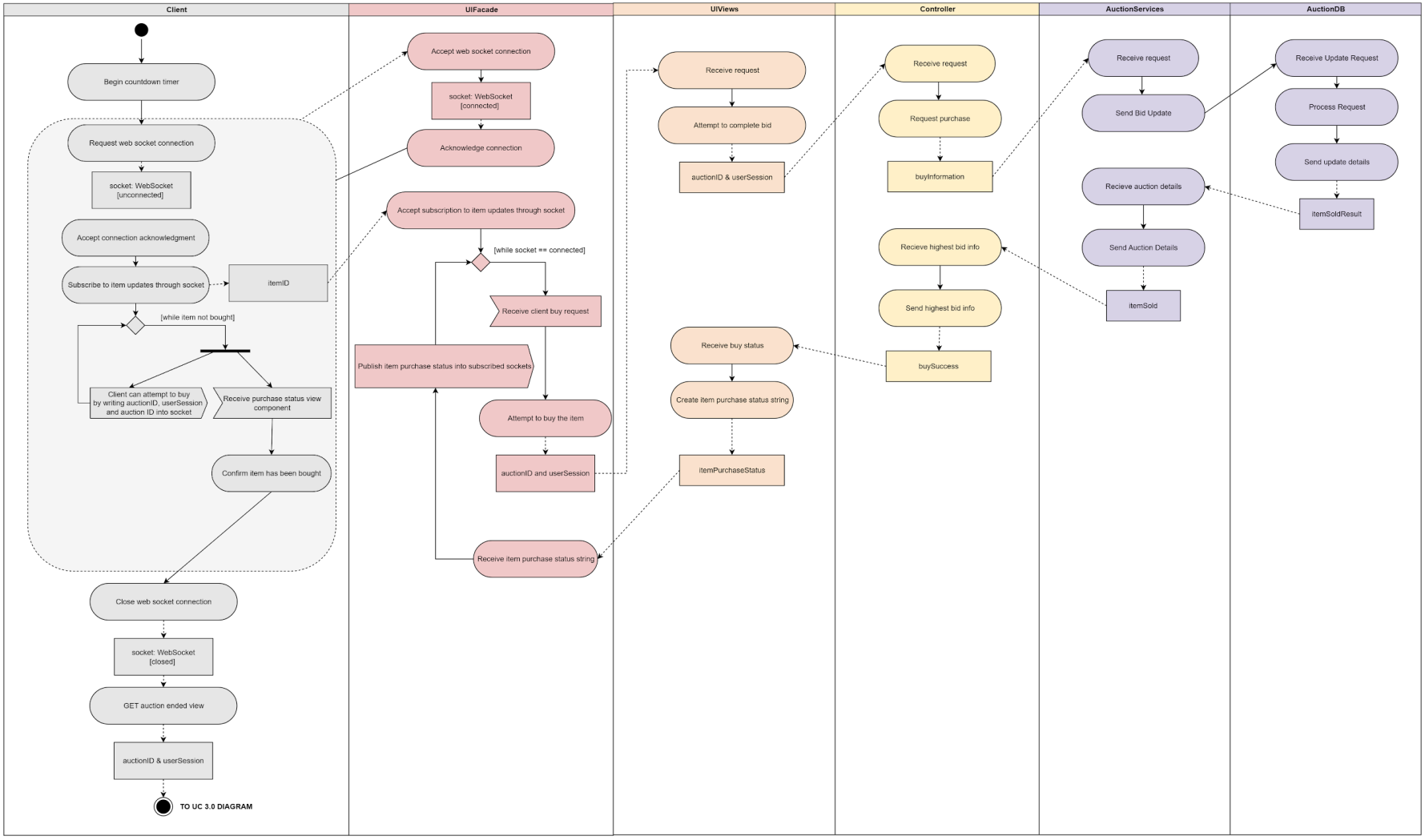


Fig.16. *UC 3.2 Dutch Auction Bidding*. Client assumed to be on *Dutch Auction Bidding UI* at the beginning of the activity. Notice that the ending activity UC3.0 is very similar to UC4.0, so refer to UC4.0 for what an activity of UC3.0 would look like. To see the sequence diagram of UC3.0 refer to **Fig.7**.

5.4 UC4: Auction Ended

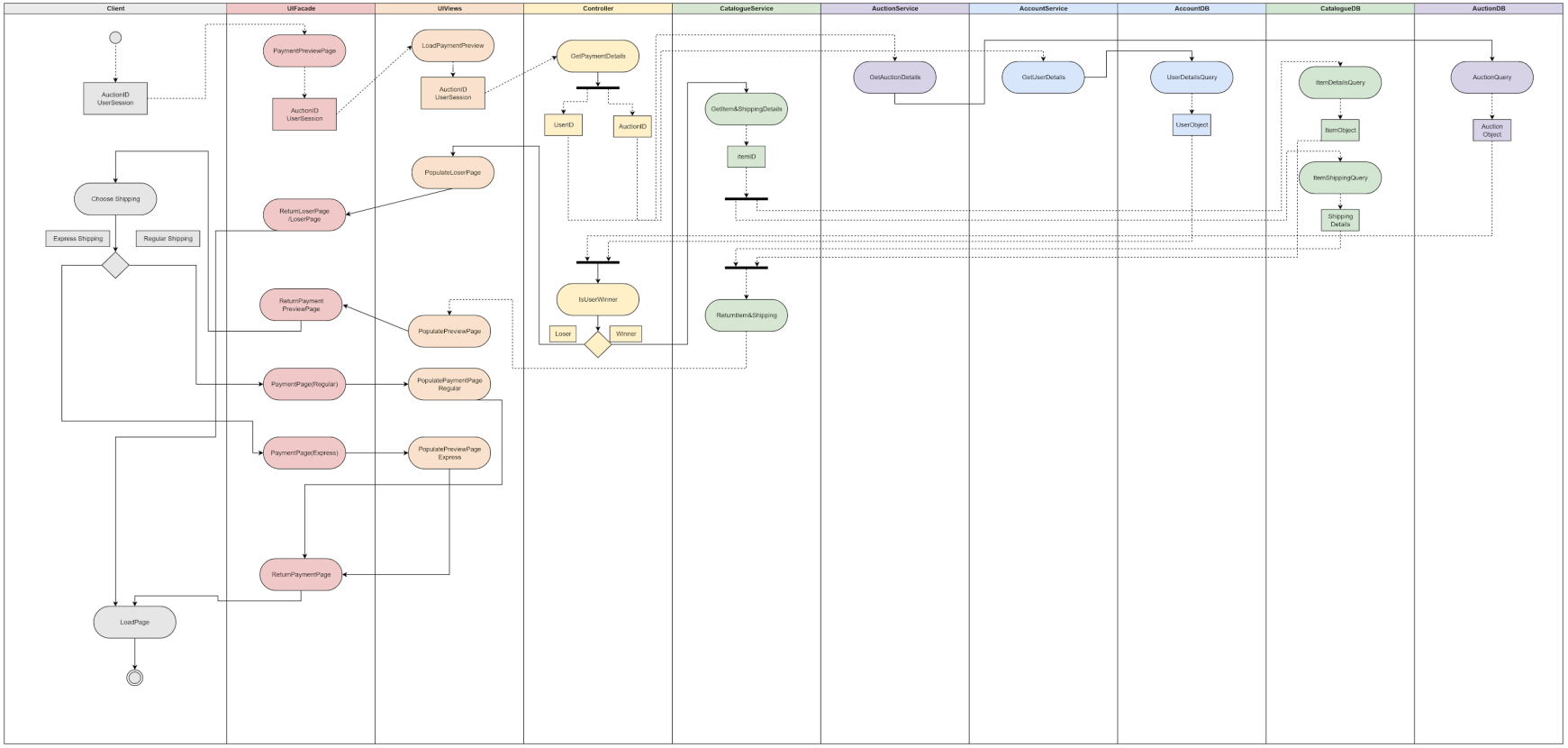


Fig.17. *UC 4.0 Auction Ended* activity. Client assumed to be on *Auction Ended UI* at beginning of activity.

5.5 UC5: Payment

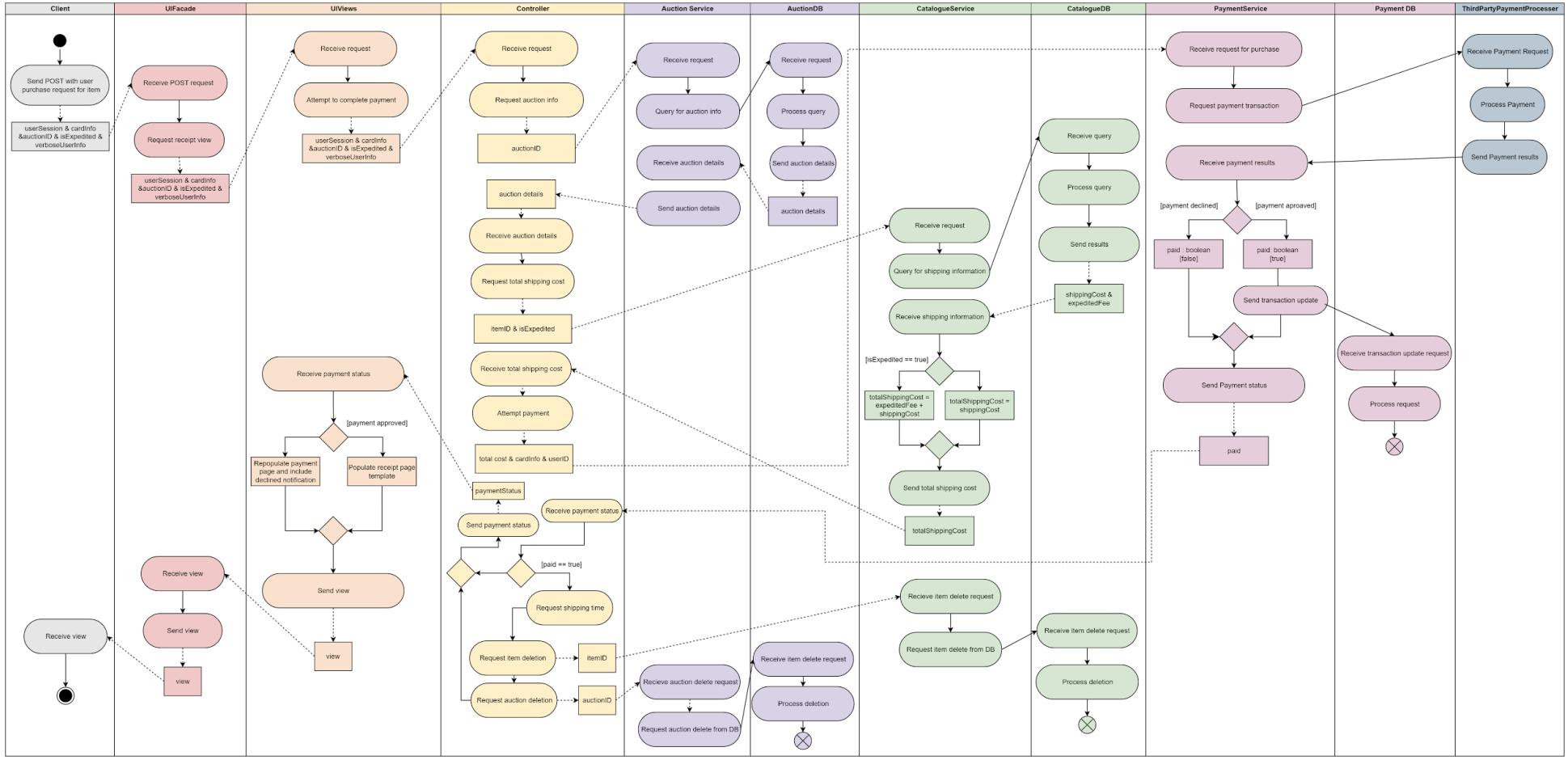


Fig.18. *UC 5.0 Payment* activity. Client assumed to be at *Payment Page UI* at beginning of activity.

5.6 UC6: Receipt Page and Shipment Details

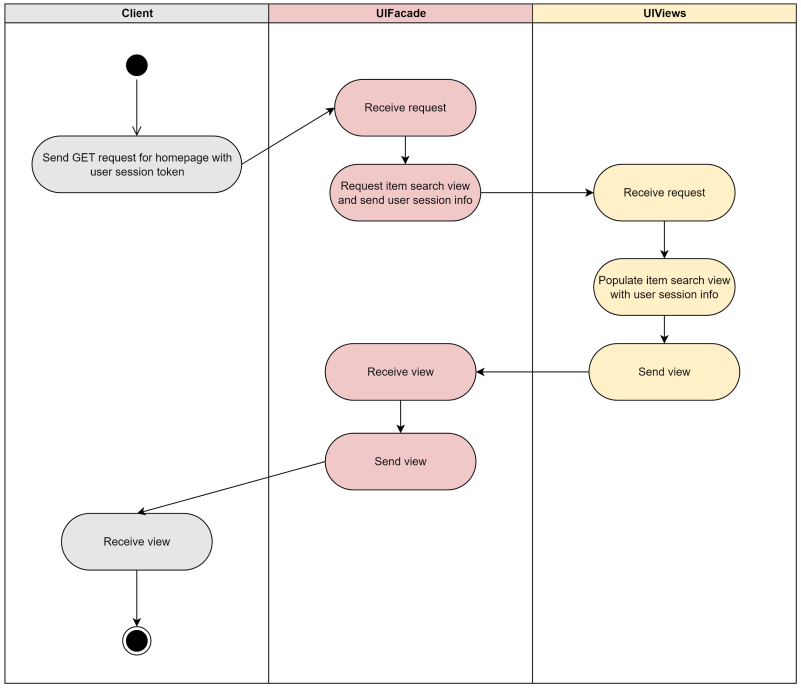
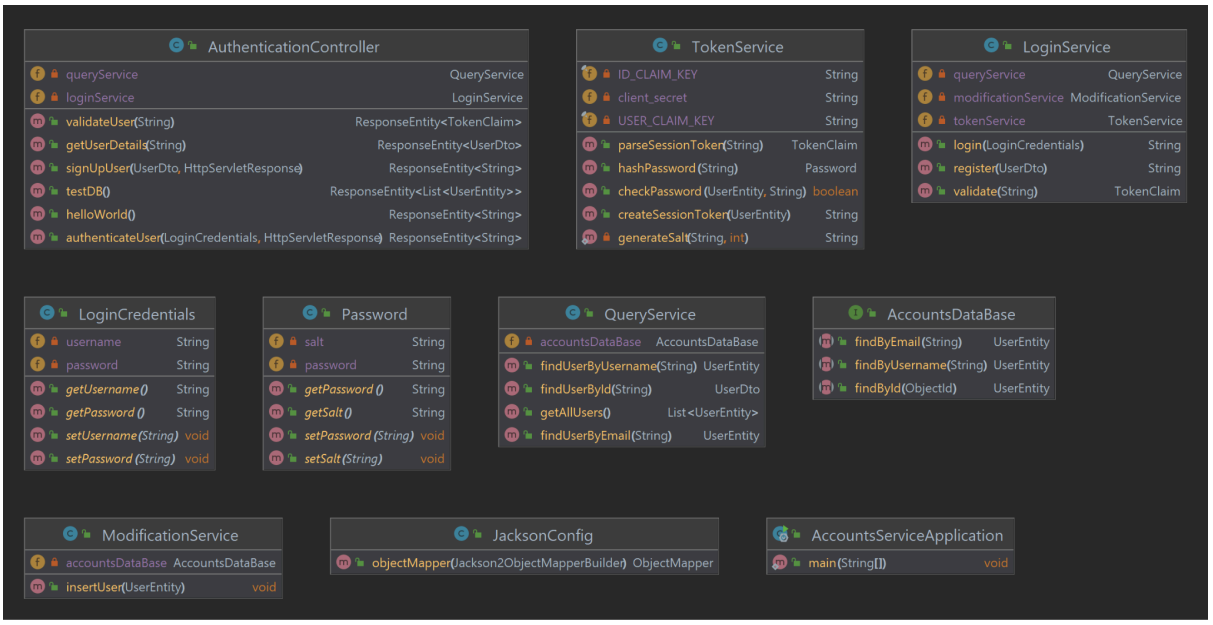
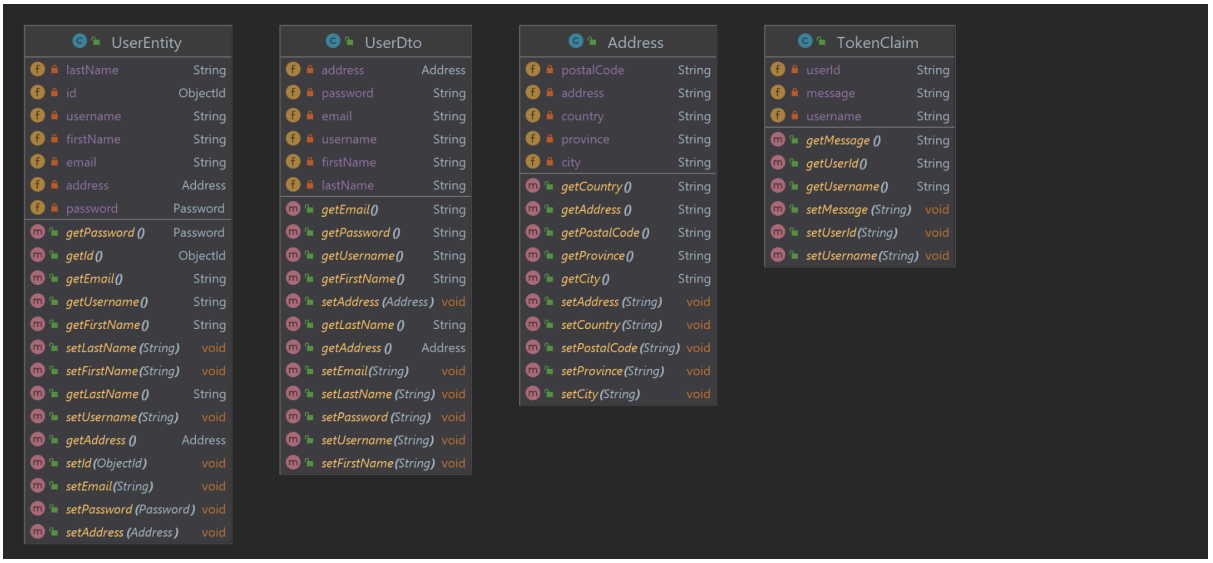


Fig. 19. *UC 5.0 Recept Page and Shipment Details* activity. **Client is assumed to be at the *Receipt and Shopping Details UI* at the beginning of sequence.** This activity thus represents a simple navigation to our homepage which we chose to be the *Item Search View*. See **Fig. 19** to confirm that the receipt page and all of its details was returned to the client as appropriate.

6 Class Diagrams

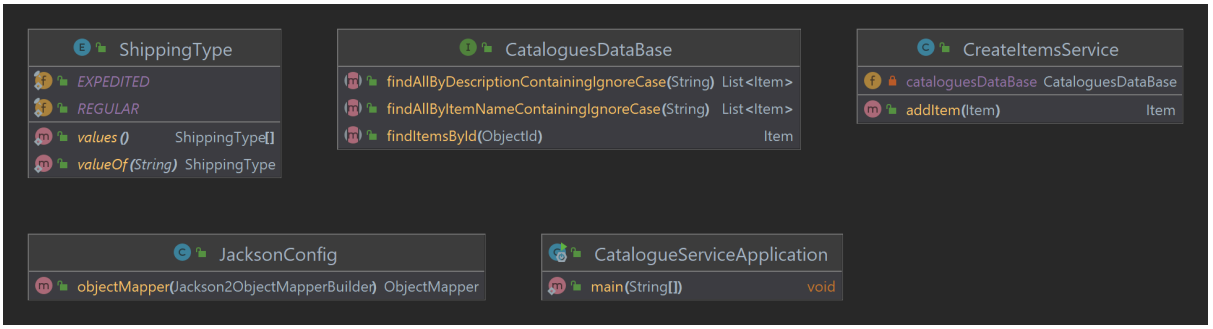
6.1 Account Service Diagrams

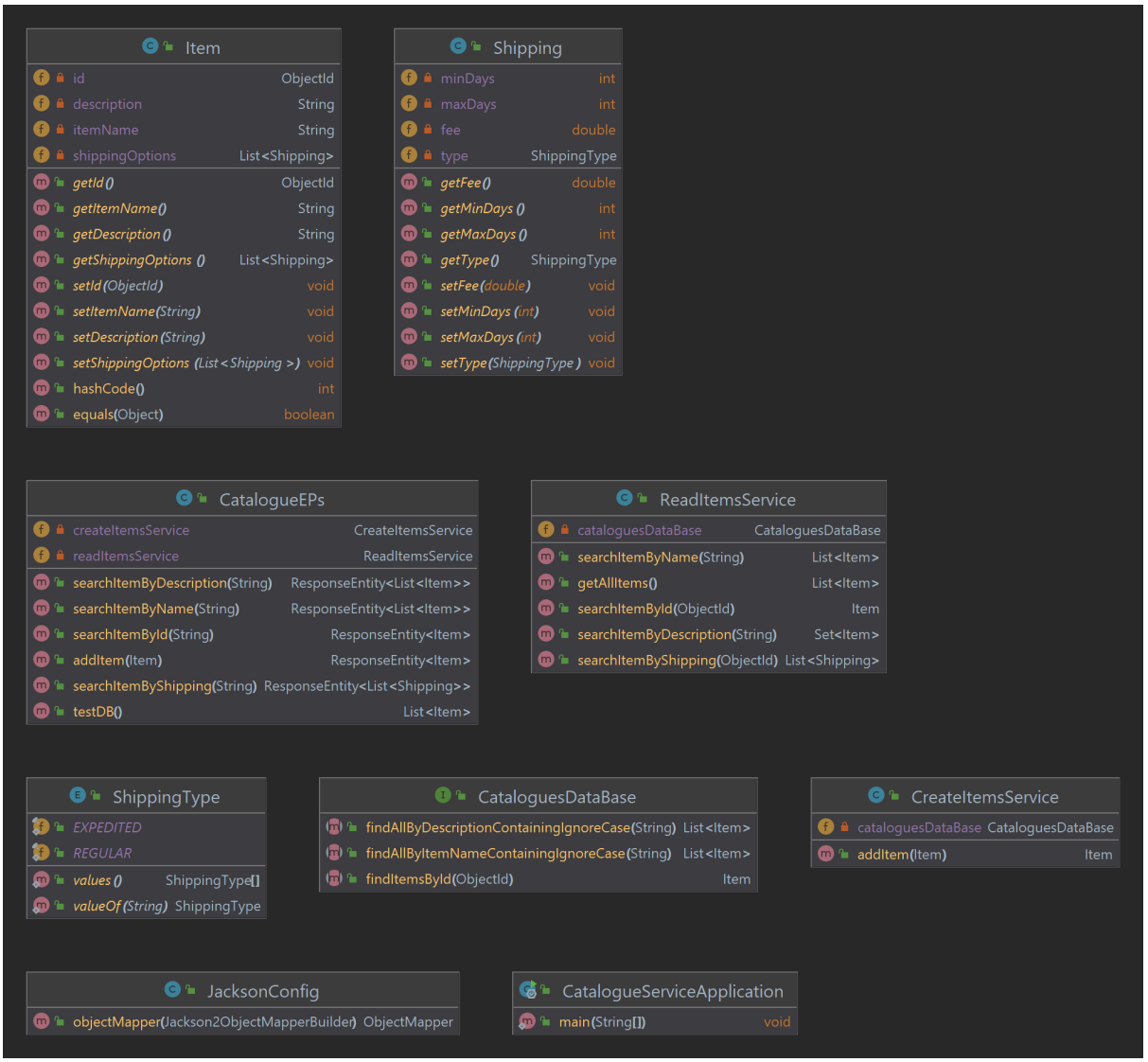




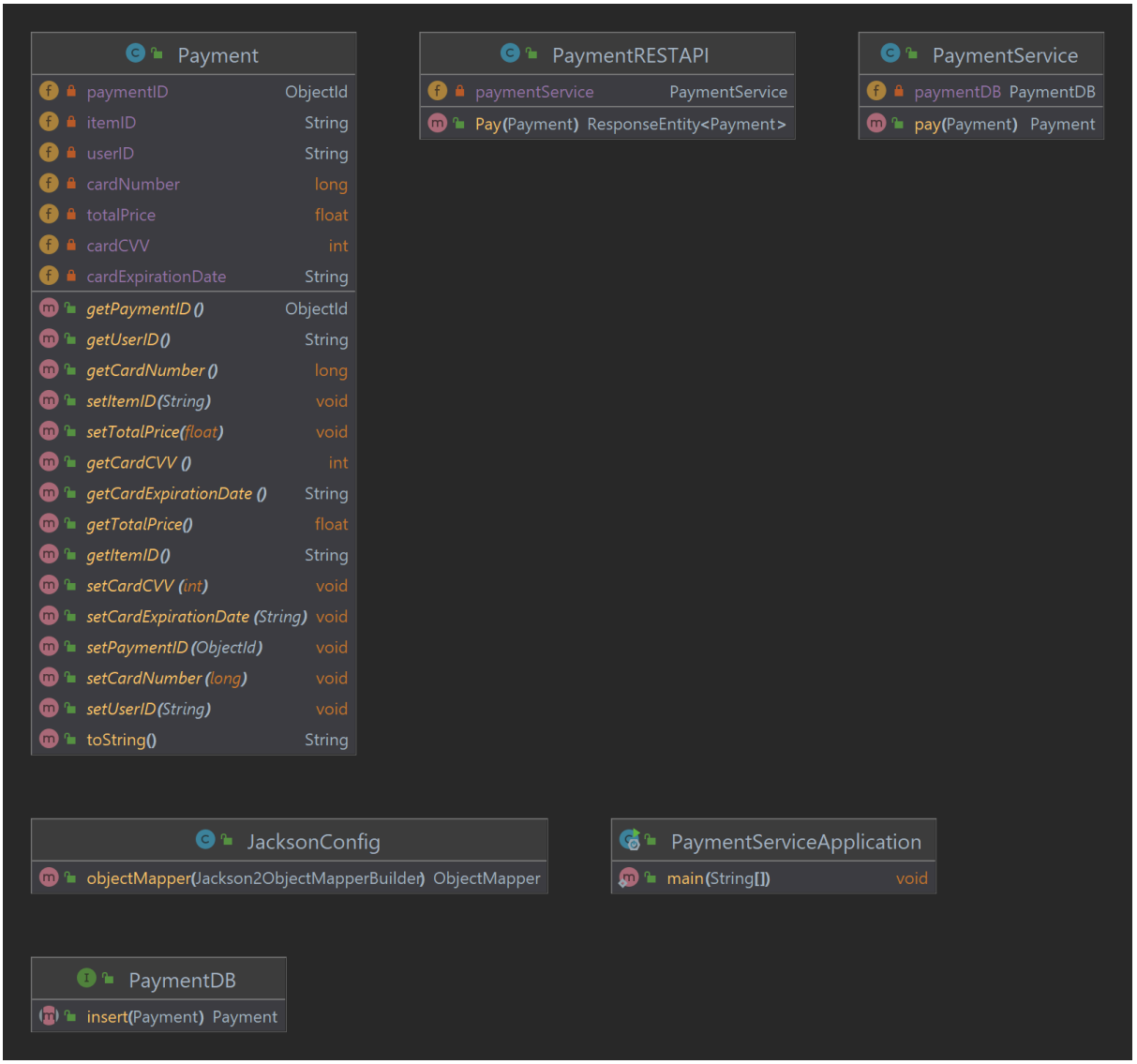
6.2 Auction Service Diagrams

6.3 Catalogue Service Diagrams





6.4 Payment Service Diagrams



7 Architecture

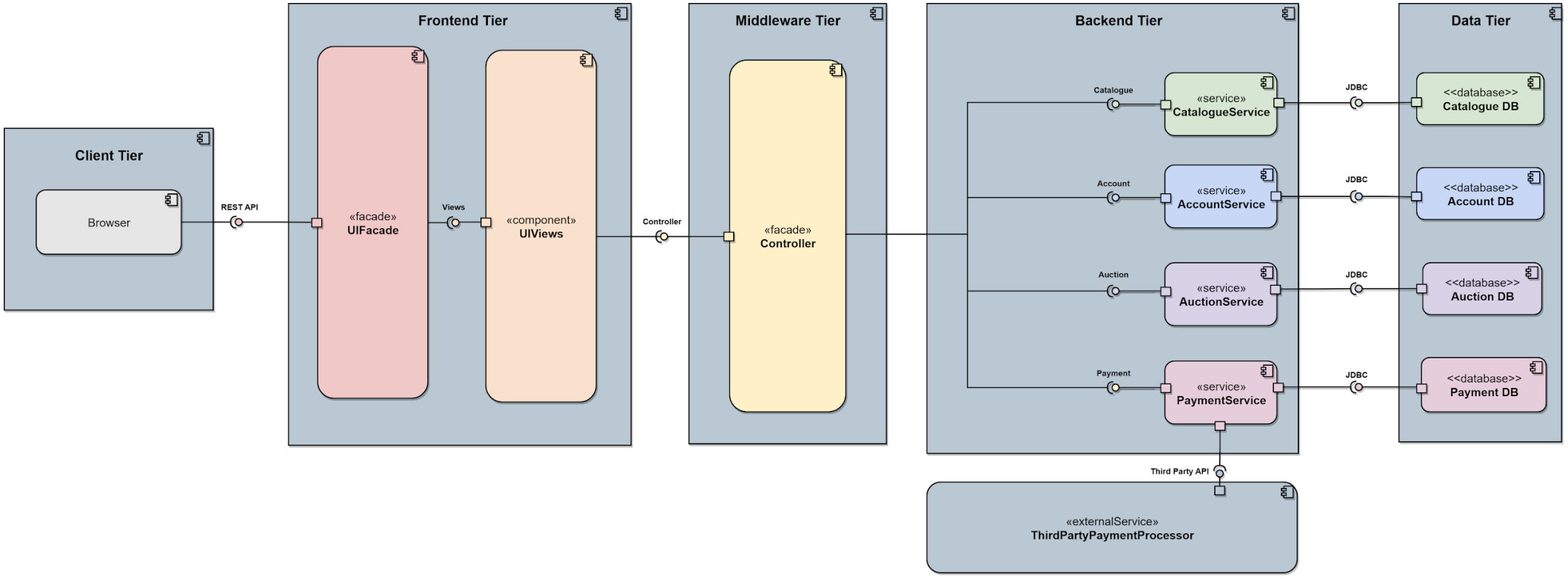


Fig 20. A UML component diagram of the initial decomposition of our system. The details of the components and their exposed interfaces can be found in the tables of section **6.1 System Modules** and section **6.2 System Interfaces**.

7.1 System Modules

Modules			
Module Name	**Description**	**Exposed Interface Names**	**Interface Description**
UIFacade	“Accepts HTTP requests from the client, calls an appropriate operation from the ViewsInterface to retrieve an appropriate view which is returned to the client.”	UIFacade:REST API	“REST API exposes a set of URLs through which clients on a browser can make HTTP requests to.”
UIViews	“UIViews is responsible for forwarding client requests to the Middleware, retrieving updated data from the Middleware, preparing a view, and returning the view.”	UIViews:Views	“Provides methods which accept parameters parsed from client HTTP requests. Every HTTP request accepted by the UIFacade has a corresponding method exposed by ViewsInterface.”
ControllerFacade	“This module acts as a facade for incoming client requests from UIViews. It distributes the work of fulfilling a client request by calling one or more services on our Backend.”	ControllerFacade: Controller	”Exposes a set of methods that fulfil client requests by making calls to the Backend.
CatalogueService	“This module is responsible for any backend requests relating to auction items in our system. It has its own database for storage of item data.”	CatalogueService: Catalogue	“Provides methods that allow for searching for and the deletion of items in our system.”
CatalogueDB	“Stores persistent data related to auction items.”	CatalogueDB: MongoDB Java Driver	“Provides methods that allow the client to perform CRUD operations on a MongoDB database.”
AccountService	“This module is responsible for any backend requests relating to user accounts. It has its own database for account data.”	AccountService: Account	“Provides methods that allow for account management including account registration and authentication.”

AccountDB	“Stores persistent data related to accounts.”	AccountDB: MongoDB Java Driver	“Provides methods that allow the client to perform CRUD operations on a MongoDB database.”
AuctionService	“This module is responsible for managing system auctions including bid and purchase updates. It communicates with its own auction database.”	AuctionService: Auction	“Provides methods that allow for dutch auction purchases, forward auction bid updates, and auction information retrieval.”
AuctionDB	“Stores persistent data related to Dutch auctions and forward auctions.”	AuctionDB: MongoDB Java Driver	“Provides methods that allow the client to perform CRUD operations on a MongoDB database.”
PaymentService	“This module is responsible for backend payment requests. It has its own database for storage of payment history.”	PaymentService: Payment	“Provides methods that allow for client payment.”
PaymentDB	“Stores persistent data related to payment history.”	PaymentDB: MongoDB Java Driver	“Provides methods that allow the client to perform CRUD operations on a MongoDB database.”

7.1 System Interfaces

Interfaces		
Interfaces	**Operations**	**Operation Description**
UIFacade: REST API	<HTTP response> REST API:GET/login, used by the client browser.	“HTTP method for retrieving login page.”
	<HTTP response> REST API:POST /auctionlist?userSession&auctionID, used by the client browser.	”HTTP method for requesting to join an auction with the passed auction ID.”
	<HTTP response> REST API:GET/login?name&pwd, used by the client browser.	“HTTP method for logging into the system.”
	<HTTP response> REST API:POST/signUp?username&address&pwd, used by the client browser.	“HTTP method for user registration”
	<HTTP response> REST API:POST /auctionList?userSession&auctionID used by the client browser.	”HTTP method for retrieving list of auctions matching item description.”
	<HTTP response> REST API:POST /auctionList?userSession&auctionID used by the client browser.	”HTTP method for retrieving list of auctions matching item description.”
	<HTML file> UIViews:Views:getLoginStatusView(UserSession : String, password : String) used by UIFacade.	“Authenticates a user via a call to the Middleware; it returns a view that corresponds to the success of authentication”

UIViews: Views	<HTML file> UIViews:Views:getAuctionStatusView() used by UIFacade.	“Asks the middleware to retrieve the item auction page; Returns a view based on what type of auction it is. If the auction has already ended it will return an updated refreshed view.”
	<HTML file> UIViews:Views:getSignUpView(signUpInfo : String) used by UIFacade.	“Attempts to add a new user to the system by a call to the Middleware; it returns a view based that corresponds to the success of registration.”
	<HTML file> UIViews:Views:getAuctionListView(itemID : String, userSession : String) used by UIFacade.	“Asks the Middleware to retrieve a list of items that match an item description. Returns to the user a view populated with a list of matched items.”
	<HTML file> UIViews:Views:getAuctionView(AuctionID:) used by UIFacade.	“Asks the Middleware to retrieve a list of items that match an item description. Returns to the user a view populated with a list of matched items.”
	<HTML file> UIViews:Views:getUpdatedBidView() used by UIFacade.	“Asks the Middleware to retrieve the updated price for an item, for a forward auction item. Returns a string with highest bid and highest bidder”
	<HTML file> UIViews:Views:getAuctionEndedView() used by UIFacade.	“Asks the Middleware to retrieve the same view for all clients to notify that the auction is over. Returns AuctionEnded view if auction is ended or a refreshed Forward/Dutch view if it is live
	<HTML file> UIViews:Views:getPaymentView() used by UIFacade.	“Asks the Middleware to retrieve a information regarding auction item price and user information. Returns a paymentView if the client is the auction winner , else a loser view.”
	<HTML file> UIViews:Views:getReceiptPage() used by UIFacade.	“Asks the Middleware to gather receiptInformation and to see if the payment went through. Returns a ReceiptPage if payment is successful otherwise refreshes the page with an error message”
	<HTML file> UIViews:Views:getItemSearchView() used by UIFacade.	“ Returns the ItemSearchView. Same view as when a client first logs in or signsup”

Controller: Controller	<details : PaymentPreviewDetails> Controller:Controller:getPaymentPreviewDetails(auctionID : String, userSession : String) used by UIViews.	“Retrieves payment preview information by calling services on the Backend..”
	<loginSuccess : boolean> Controller:Controller:login(userID : String, password : String) used by UIViews.	“Communicates with services on the Backend to determine if a user exists; returns true if yes, else false”
	<signUpStatus : boolean> Controller:Controller:signUp(userInfo : String) used by UIViews.	“Communicates with services on the Backend to register a user. Returns true if successful, else false.”
	<purchaseStatus : boolean> Controller:Controller:attemptBuy(auctionID : String, userSession : String) used by UIViews.	“Attempts to purchase an item up for auction (verified to be of type Dutch in Backend) by making calls to the services on the Backend. Returns true if successful, else false.”
	<auctionEnded : boolean> Controller:Controller:getAuctionStatus(auctionID : String) used by UIViews.	“Communicates with services on the Backend to determine if the auction of an item with the itemID has ended or not; returns true if yes, else false”
	<highestBidInfo : String> Controller:Controller:bidForItem(auctionID : String, userSession : String, amount : String) used by UIViews.	“Attempts to update the bid of an item to the amount String by calling services on the Backend. The bid information of the highest bid after the update attempt is returned”
	<auction : Auction> Controller:Controller:getAuction(auctionID : String) used by UIViews.	“Communicates with services on the Backend to retrieve the latest information on an auction.”
	<auctions : List<Auction>> Controller:Controller:getAuctionList(itemDescription : String, userSession : String) used by UIViews.	“Communicates with services on the Backend to retrieve auction objects that match the passed description”
	<paymentSuccess : boolean> Controller:Controller:processPayment(cardInfo : String, userSession : String) used by UIViews.	“Communicates with services on the Backend to complete an auction winner’s item purchase. It returns true if payment is successful.”
	<items:List<Item>> CatalogueService:Catalogue: findItemByDescription(ItemDescription: String) used by Controller	“This operation allows the Controller to communicate with the CatalogueService on the Backend and retrieve a list of Item objects that match the provided item description.”

CatalogueService: Catalogue	<ItemDetails:Item> CatalogueService:Catalogue: findItemById(ItemID: String) used by Controller	“This operation allows the Controller to communicate with the CatalogueService on the Backend and retrieve details of an Item object that matches the provided ItemID.”
	<ShippingDetails:String> CatalogueService:Catalogue: findItemShipping(ItemID: String) used by Controller	“This operation allows the Controller to communicate with the CatalogueService on the Backend and retrieve the shipping details of an Item object that matches the provided ItemID.”
	<items:Item> CatalogueService:Catalogue: findItemByName(ItemName: String) used by Controller	“This operation allows the Controller to communicate with the CatalogueService on the Backend and retrieve an Item object that matches the provided ItemName.”
	<itemAvailable:boolean> CatalogueService:Catalogue: isItemSold(ItemID: String) used by Controller	“This operation allows the Controller to communicate with the CatalogueService on the Backend and determine if the Item object that matches the provided ItemID has been sold or is still available for purchase. The interface returns a boolean value indicating the availability of the item.”
AccountService: Account	<userValid:boolean> AccountService:Account: isUserSessionValid(userToken: JSON) used by Controller	“This operation allows the Controller to communicate with the AccountService on the Backend and verify the validity of the provided userToken. The interface returns a boolean value indicating whether the user session is still valid or has expired.”
	<userSession:JSON-Token> AccountService:Account: authenticateUser(userName: String, password:String) used by Controller	“This operation allows the Controller to communicate with the AccountService on the Backend and authenticate a user with the provided userName and password. The interface returns a JSON token that represents the user session.”
	<userSession:JSON-Token> AccountService:Account: signUpUser(userDetails: JSON) used by Controller	“This operation allows the Controller to communicate with the AccountService on the Backend and create a new user account using the provided userDetails in JSON format. The interface returns a JSON token that represents the user session.”

	<userSession:JSON-Token> AccountService:Account: signUpUser(userDetails: JSON) used by Controller	“This operation allows the Controller to communicate with the AccountService on the Backend and update the user details using the provided userDetails in JSON format. The interface returns a JSON token that represents the updated user session.”
AuctionService: Auctions	<auctionDetails: Auction> AuctionService:Auctions: dutchAuctionPurchase(AuctionID: String,paymentAmount: int, UserID: String) used by Controller	“This operation allows the Controller to communicate with the AuctionService on the Backend and purchase an auction item using the Dutch Auction method. The operation requires an AuctionID, paymentAmount, and UserID to complete the purchase and returns auctionDetails of the purchased item.”
	<auctionDetails:Auction> AuctionService:Auctions: findAuctionByID(AuctionID: String) used by Controller	“This operation allows the Controller to communicate with the AuctionService on the Backend and retrieve auctionDetails of an Auction object that matches the provided AuctionID.”
	<auction:Auction> AuctionService:Auctions: findAuctionByItem(ItemID: String) used by Controller	“This operation allows the Controller to communicate with the AuctionService on the Backend and retrieve an Auction object that matches the provided ItemID.”
	<auctionFinished:boolean> AuctionService:Auctions: auctionEnded(AuctionID: String) used by Controller	“This operation allows the Controller to communicate with the AuctionService on the Backend and check if the Auction object that matches the provided AuctionID has ended. The operation returns a boolean value indicating whether the auction has finished or not.”
	<isCheckedOut:boolean> AuctionService:Auctions: isAuctionInCheckout(AuctionID: String) used by Controller	“This operation allows the Controller to communicate with the AuctionService on the Backend and check if the Auction object that matches the provided AuctionID is in the checkout process. The operation returns a boolean value indicating whether the auction is in checkout or not.”
	<getHighestBid:JSON> AuctionService:Auctions: getAuctionHighestBid(AuctionID: String) used by Controller	“This operation allows the Controller to communicate with the AuctionService on the Backend and retrieve the highest bid for the Auction object that matches the provided AuctionID. The operation returns the highest bid value in JSON format.”

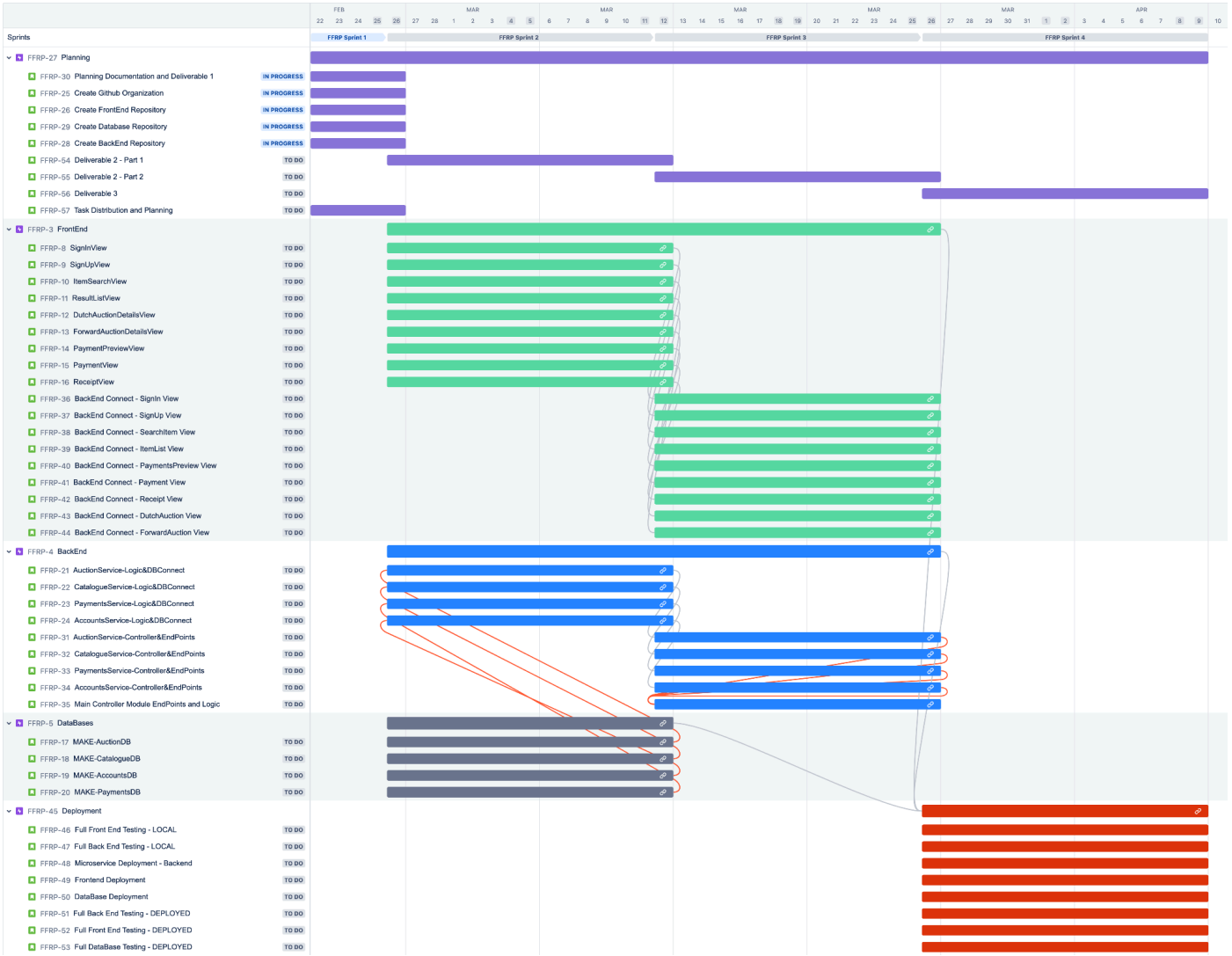
	<auctionType:String> AuctionService:Auctions: getAuctionType(AuctionID: String) used by Controller	"This operation allows the Controller to communicate with the AuctionService on the Backend and retrieve the auction type for the Auction object that matches the provided AuctionID. The operation returns the auction type value in string format"
	<auctionEndResult:boolean> AuctionService:Auctions: closeAuction(AuctionID: String) used by Controller	"This operation allows the Controller to communicate with the AuctionService on the Backend and close an Auction object that matches the provided AuctionID. The operation returns a boolean value indicating whether the auction closure was successful or not."
	<auctionDetails:Auction> AuctionService:Auctions: forwardNewBid(AuctionID: String, bidAmount: int, UserID: String) used by Controller	"This operation allows the Controller to communicate with the AuctionService on the Backend and place a new bid on an Auction object that matches the provided AuctionID. The operation requires the AuctionID, bidAmount, and UserID to place a bid and returns updated auctionDetails of the Auction object with the new bid information."
PaymentServices: Payment	<purchaseDetails: Purchase> PaymentServices:Payment: payTotalAmount(amount: int, UserID: String, paymentInfo: JSON) used by Controller	"Communicates with Payment Services to process payment for the purchase made by the user identified by the given UserID, with the total purchase amount specified in the amount parameter and the payment information provided as a JSON object in the paymentInfo parameter. Returns the details of the purchase made in the purchaseDetails parameter."
	<purchaseDetails: Purchase> PaymentServices:Payment: getReceiptInfo(purchaseId: String, UserID: String) used by Controller	"Communicates with Payment Services to retrieve the receipt information for the purchase identified by the given purchaseId and made by the user identified by the given UserID. Returns the details of the purchase in the purchaseDetails parameter."
MongoDB Java Driver	For API operation list and details refer to: MongoDB Team. "MongoDB Java Driver". *MongoDB Website*, https://www.mongodb.com/docs/drivers/java/sync/current accessed on 2/27/2023.	

8 Activity Plan

Task	Section	Sprint
SignInView	FrontEnd	2
SignUpView	FrontEnd	2
ItemSearchView	FrontEnd	2
ItemListView	FrontEnd	2
DutchAuctionDetailsView	FrontEnd	2
ForwardAuctionDetailsView	FrontEnd	2
PaymentPreviewView	FrontEnd	2
PaymentView	FrontEnd	2
ReceiptView	FrontEnd	2
MAKE-AuctionDB	DataBase	2
MAKE-CatalogueDB	DataBase	2
MAKE-AccountsDB	DataBase	2
MAKE-PaymentsDB	DataBase	2
AuctionService-Logic&DBConnect	BackEnd	2
CatalogueService-Logic&DBConnect	BackEnd	2
PaymentsService-Logic&DBConnect	BackEnd	2
AccountsService-Logic&DBConnect	BackEnd	2
AuctionService-Controller&EndPoints	BackEnd	3
CatalogueService-Controller&EndPoints	BackEnd	3
PaymentsService-Controller&EndPoints	BackEnd	3
AccountsService-Controller&EndPoints	BackEnd	3
Main Controller Module EndPoints and Logic	BackEnd	3
BackEnd Connect - SignIn View	FrontEnd	3
BackEnd Connect - SignUp View	FrontEnd	3

BackEnd Connect - SearchItem View	FrontEnd	3
BackEnd Connect - ItemList View	FrontEnd	3
BackEnd Connect - PaymentsPreview View	FrontEnd	3
BackEnd Connect - PaymentsPreview View	FrontEnd	3
BackEnd Connect - Payment View	FrontEnd	3
BackEnd Connect - Receipt View	FrontEnd	3
BackEnd Connect - DutchAuction View	FrontEnd	3
BackEnd Connect - ForwardAuction View	FrontEnd	3
Full Front End Testing - LOCAL	Deployment	4
Full Back End Testing - LOCAL	Deployment	4
Microservice Deployment - Backend	Deployment	4
Frontend Deployment	Deployment	4
DataBase Deployment	Deployment	4
Full Back End Testing - DEPLOYED	Deployment	4
Full Front End Testing - DEPLOYED	Deployment	4
Full DataBase Testing - DEPLOYED	Deployment	4

8.1 Project Backlog and Sprint Backlog



8.2 Group Meeting Logs

Present Group Members	Meeting Date	Issues Discussed/Resolved
Pouya Sameni Shalom Asbell Adit Jain Sana Khademi	02/06/2023	1. Breaking up tasks between each member - Pouya & Sana: creating diagrams for the even numbered use cases - Shalom & Adit: creating diagrams for the odd numbered use cases 2. Discussed the overall design and architecture of how we want to build the system 3. Reviewed the options for the tech stack we all want to use to build this system (some of the options were): - Java Spring - React Js - MongoDB
Pouya Sameni Shalom Asbell Adit Jain Sana Khademi	02/10/2023	1. Went over everyones progress on the tasks assigned (creating the sequence diagrams, activity diagrams and UML) - Reviewed and discussed each other diagrams and how we can improve and went over any clarifications 2. Decided a date to have the diagrams completed by so that we can work on the design document - Goal: finish diagrams by: 02/12/2023
Pouya Sameni Shalom Asbell Adit Jain Sana Khademi	02/22/2023	1. Redefined our UML diagram resulting in a re-work in how we create our activity and system diagrams 2. Created System diagrams and started report
Pouya Sameni Shalom Asbell Adit Jain Sana Khademi	02/25/2023	1. Reviewed all of the sections of the report and diagrams 2. Made final touches and revisions

9 Test Driven Development

Test cases will be provided in the form of a table as follows:
Note the Test ID : is in the format “Use case: Use case subsection, ID”
Example: Test ID : 1.1.0 is equivalent to saying Use case 1 , sign up , Test 0

Test ID:	1.1.0 (1)
Category:	addition of a user in Account DB
Requirements Coverage:	One more unique user in Account DB
Initial Condition:	DB has already been initiated and is live
Procedure:	*1. The user selects signUp* *2. The user provides firstName, LastName, address, UserName, password* *3. User Clicks Submit* *4. User is directed to the item search page*
Expected Outcome:	AccountDB has a new row with UserName as a unique key along with columns with respective values
Notes:	

Test ID:	1.2.0 (2)
Category:	ULI- User Log in
Requirements Coverage:	ULI - User Login with correct credentials
Initial Condition:	User exists in the Accounts DB
Procedure:	*1. The user selects signIn* *2. User inputs correct UserName and password* *3. User clicks SignIn* *4. User is directed to the item search page*
Expected Outcome:	A call to the AcountDB that confirms if key-value pair of UserName and password exists
Notes:	

Test ID:	1.2.1 (3)
Category:	ULI- User Login
Requirements Coverage:	ULI - User Login with Incorrect credentials
Initial Condition:	-Username in the Accounts DB - accounts microservice is live
Procedure:	*1. The user selects signIn* *2. User inputs UserName and Incorrect password* *3. User clicks SignIn* *4. User is directed to the item failedLogIn page*
Expected Outcome:	Microserve accountService returns false for verify()
Notes:	

Test ID:	1.2.2 (4)
Category:	ULI- User Log in
Requirements Coverage:	ULI - User Login with Incorrect credentials
Initial Condition:	-User-name isnt in the Account DB
Procedure:	*1. The user selects signIn* *2. User inputs UserName and Incorrect password* *3. User clicks SignIn* *4. User is directed to the item failedLogIn page*
Expected Outcome:	Microserve accountService returns false for verify()
Notes:	

Test ID:	3.1.0 (5)
Category:	FA-Forward Auction
Requirements Coverage:	FA- Underbid/sameBid
Initial Condition:	-Item is for auction - item exists in a non ‘sold’ state in AuctionDB -item is in a forward auction
Procedure:	*1. User is on the item Page* *2. User inputs value lower/same bid as the highest current bid into input textbox and clicks bid*
Expected Outcome:	User has a refreshed page with highest current bid and highest current bid value
Notes:	*The refreshed page has a new/same higher bid from the auctionDB. The new bid value cannot be the User bid*

Test ID:	3.1.1 (6)
Category:	FA-Forward Auction
Requirements Coverage:	FA- HigherBid
Initial Condition:	-Item is for auction - item exists in a non ‘sold’ state in AuctionDB -item is in a forward auction -timeleft > 0
Procedure:	*1. User is on the item Page* *2. User inputs value a larger bid as the highest current bid into input textbox and clicks bid*
Expected Outcome:	User has a refreshed page with the highest current bid and highest current bid value.
Notes:	The refreshed page may not show the User as the highest bid since there may be a new bidder

Test ID:	3.1.2 (7)
Category:	FA-Forward Auction
Requirements Coverage:	FA- AuctionEnded
Initial Condition:	-Item is for auction - item exists in a non ‘sold’ state in AuctionDB - item is in a forward auction -User does not have the highest bid
Procedure:	*1. User is on the Item Page* *2. Time decrement results in timeLeft = 0* *3. Page gets re-rendered with the highest bidder ‘UserName’ stated as the winner.* *4. Button for ‘Pay for item’*
Expected Outcome:	Results now with Pay now View with winner name
Notes:	The refreshed page may not show the User as the highest bid since there may be a new bidder

Test ID:	3.2.0 (8)
Category:	DA- Dutch Auction
Requirements Coverage:	DA- Buy Item
Initial Condition:	- Item is for auction - Price of item > 0 - Item exists in a non ‘sold’ state in AuctionDB - Item is in a dutch auction state
Procedure:	*1. User is on the Item Page* *2. User clicks buy now*
Expected Outcome:	-Returns Pay now page view with winner Username requesting shipping information.
Notes:	This is based on the assumption that when a user clicks buy now, They are responsable for purchasing the item.

Test ID:	4.0.0 (9)
Category:	SP - Shipping Page
Requirements Coverage:	SP - Calculating total price with regular shipping
Initial Condition:	- Item exists in a ‘sold’ state in AuctionDB - Item exists in the Catalogue DB with shipping price - Item is in a sold state - Shipping preference is not selected
Procedure:	*1. User gets redirected to the Pay Now Page* *2. User selects Regular shipping* *3. User selects pay now*
Expected Outcome:	- User gets redirected to Payment detail page with correct shipping details as follows: Shipping price= Price of item + Regular Shipping
Notes:	This is based on the assumption that when a user clicks buy now, They are responsable for purchasing the item.

Test ID:	4.0.1 (10)
Category:	SP - Shipping Page
Requirements Coverage:	SP - Calculating total price with expedited shipping
Initial Condition:	- Item exists in a ‘sold’ state in AuctionDB - Item exists in the Catalogue DB with shipping price and expedited shipping price - Item is in a sold state - Shipping preference is not selected
Procedure:	*1. User getes redirected to the Pay Now Page* *2. User selects Expidited shipping* *3. User selects pay now*
Expected Outcome:	- User gets redirected to Payment detail page with correct shipping details as follows: Shipping price= Price of item + Expedited Fee + Shipping cost
Notes:	This is based on the assumption that when a user clicks buy now, They are responsable for purchasing the item.

Test ID:	5.0.0 (11)
Category:	PP - Payment Page
Requirements Coverage:	PP - User pays for their purchase using credit card
Initial Condition:	- User is the winner of the auction - The auction has ended or if dutch auction the user is the first to click pay - User will follow through with the payment
Procedure:	*1. User gets redirected to the Payment Page* *2. User enters their credit card information* *3. User selects pay now*
Expected Outcome:	- The backend connects to payment service to conduct a payment for the amount - The payment is approved - purchase is stored in the database - receipt and the purchase ID is returned
Notes:	This is based on the assumption that when a user clicks buy now, They are responsible for purchasing the item.

Test ID:	2.1.0 (12)
Category:	ISP- Item Search Page
Requirements Coverage:	ISP- User searches for items using description or name
Initial Condition:	- User is logged in successfully - Items are available to search for in the database
Procedure:	*1. User searches for the item they are looking for* *2. User clicks the search button and waits*
Expected Outcome:	- The system should talk to the backend and find matching item names and descriptions - A list of active auctions is returned for user to browse
Notes:	

Test ID:	2.3.0 (13)
Category:	ALP- Auction List Page
Requirements Coverage:	ALP - User Selects a auction to view details of
Initial Condition:	- User is logged in successfully - Item search has already been made - there are available auctions on the item list
Procedure:	*1. User selects an auction using a radio button*
Expected Outcome:	- The system should talk to the backend and find matching auction find its descriptions and details - A websocket should be opened and a detailed page should be loaded for the user with all the auction details
Notes:	

Test ID:	6.0.0 (14)
Category:	RVP - Receipt View Page
Requirements Coverage:	RVP - User sees the purchase details after payment
Initial Condition:	- User is logged in successfully - User has paid for an item - Auction has ended
Procedure:	*1. User is automatically directed to this page*
Expected Outcome:	- The page redirects the user after successful payment to the receipt view - The system pulls the purchase details from the backend and displays a confirmation page with total and shipping details
Notes:	

Test ID:	1.1.0 (15)
Category:	SUP- Sign Up Page
Requirements Coverage:	SUP- User signs up for an account but uses existing credentials
Initial Condition:	- User with this credentials exists
Procedure:	*1. User enters credentials into the signup for,* *2. User clicks sign up*

Expected Outcome:	- The system should check for unique credentials - System should realise the account exists - An error should be displayed notifying the user
Notes:	

Test ID:	1.1.0 (15)
Category:	SUP- Sign Up Page
Requirements Coverage:	SUP- User signs up for an account but uses existing credentials
Initial Condition:	- User with this credentials exists
Procedure:	*1. User enters credentials into the signup for,* *2. User clicks sign up*
Expected Outcome:	- The system should check for unique credentials - System should realise the account exists - An error should be displayed notifying the user
Notes:	

Test ID:	4.0.2(16)
Category:	PPP - Payment Preview Page
Requirements Coverage:	PPP - User tries to pay for an auction they did not win
Initial Condition:	- Auction ended - User clicked on pay now button
Procedure:	*1. User is waiting for the paynow page to select shipping*
Expected Outcome:	- The system should get auction details - The system should get user details - The system should realize that the user is not the winner - The system should redirect the user to a page information customer they did not win the auction
Notes:	