Distributed Systems Report

1. Introduction:

In the Introduction, we will briefly go through the different functionalities and constraints of the Auction System.

1.1. High-level rules & information for the Auction System:

- Users in the Auction System can act as Buyers and Sellers at the same time, this implies that users' accounts can be used to act as Buyers and Sellers
- Users can login using their email and password
- The system has 3 hard-coded fake users just for quick use for testing the system
    - user 1: (sofa.joe10@yahoo.com, password)
    - user 2: (yudai@email.com, yudai_password)
    - user 3: (deven@email.com, deven_passwor)
- Users can sign up to the system given that they use an email that is not registered already
- Every Item in the Auction System belongs to a specific user
- Every user in the system will have 2 hard-coded items for quick use in testing the system
- Users can get information about the items that you have created only
- Users can create items
- Forward & Reverse Auction act as the same entity and from now onwards we will call them Normal Auctions
- Every Normal Auction in the System belongs to a specific user
- Users can Create Normal auctions and use items that they have created only for creating the normal auctions. The reserve price in normal auctions is the minimum price a seller would be willing to accept from a buyer.
- Users can close only normal auctions that they have created. If the starting price is lower than the reserve price a message will be displayed to the user stating that the item is not sold
- Users can view all the normal auctions that are currently open in the System. Users will be able to view the reserve price when viewing normal auctions if he was the creator of that normal auction
- Users can place a bid on normal auctions. Users cannot bid on an auction that they have created themselves and they cannot bid with a price that is lower than the current highest bid
- Users can view all auctions selling a specific item, this functionality is generally abbreviated as reverse auction
- Users can view all available double auctions in the system. When viewing double auctions you will be able to view if the double auction is currently running or if it has finished
- Users can view information on a single double auction by using the double auction ID

- Users can create double auctions and use items that they have created only for creating the double auctions. The user also has to specify the number of buyers and sellers for this double auction
- Users can place a buying price on a double auction only if they have not previously placed a buying/selling price on this double auction, and there are still more buyers remaining in this double auction
- Users can place a selling price on a double auction only if they have not previously placed a buying/selling price on this double auction, and there are still more sellers remaining in this double auction

1.2. Interface provided by the Auction System Server:

a. login

method: public UserInfo login(String email, String password) throws RemoteException
purpose: allows clients to log in to the Auction System using their email and password
exceptions: user enters invalid credentials

b. signup

method:public UserInfo signup(String username, String name, String email, String password) throws RemoteException
purpose: allows clients to sign up for creating new accounts in the Auction System
exceptions: user signups with an email that is already registered

c. get items specs

method: public AuctionItem getItemSpec(String email, String password, int itemId) throws RemoteException
purpose: allows clients to view information about items that they have using the itemId
exceptions: user specifies an item ID for an item that he doesn't have

d. create item

method: public int createAuctionItem(String email, String password, String itemTitle, String itemDescription, boolean itemCondition) throws RemoteException
purpose: create an item
exceptions: n/a

e. get items

method: public ArrayList<AuctionItem> getAuctionItems(String email, String password) throws RemoteException
purpose: allows clients to get all the items they own
exceptions: n/a

f. create auction

method: public int createAuction(String email, String password, int itemId, int startingPrice, String description, int reservePrice) throws RemoteException
purpose: create a normal auction by specifying:
- Item ID
- Auction Description
- starting price
- reserve price (minimum price a seller would be willing to accept from a buyer)

exceptions:
- having a starting price or a reserve price that is less than or equal to zero
- Specifying an item ID for an item that you don't have

g. close auction

method: public Auction closeAuction(String email, String password, int auctionId) throws RemoteException

purpose: allows clients to close auction
exceptions: specifying an auction ID for an auction that the client doesn't have

h. get all auctions

method: public ArrayList<Auction> getAuctions(String email, String password) throws RemoteException
purpose: get all active normal auctions in the system
exceptions: n/a

i. bid

method: public String bid(String email, String password, int auctionId, int biddingPrice) throws RemoteException
purpose: to place a bid on a normal auction
exceptions:

- using an auction ID that doesn't exist in the system
- using a negative or zero bidding price
- using a bidding price that is less than the highest current bid for this auction

j. get all auctions with a specific item name (reverse auction)

method: public ArrayList<Auction> getAuctions(String email, String password, String itemName) throws RemoteException
purpose: allows clients to browse all active normal auctions with the specific item name (also called reverse auction)
exceptions: n/a

k. get all double auctions

method: public ArrayList<DoubleAuction> getDoubleAuctions(String email, String password) throws RemoteException
purpose: get all double auctions in the system
exceptions: n/a

l. get double auction

method: public DoubleAuction getDoubleAuction(String email, String password, int auctionId) throws RemoteException
purpose: allows clients to view information about a single double auction using the double auction ID
exceptions: specifying a double auction ID that doesn't exist in the system

m. create double auction

method: public int createDoubleAuction(String email, String password, int itemId, String description, int buyersSize, int sellersSize) throws RemoteException
purpose: create a double auction by specifying:

- Item ID
- Auction Description
- number of buyers
- number of sellers

exceptions:

- Specifying an item ID for an item that you don't have
- using a negative number of buyers
- using a negative number of sellers

n. buy double auction

method: public String buyDoubleAuction(String email, String password, int auctionId, int buyingPrice) throws RemoteException
purpose: add a buying price to a double auction
exceptions:

- using a double auction ID that doesn't exist in the system

- same user is trying to place another buying/selling price on the same double auction
- using a negative number for the buying price
- trying to place a buying price on a double auction that has no more buyers

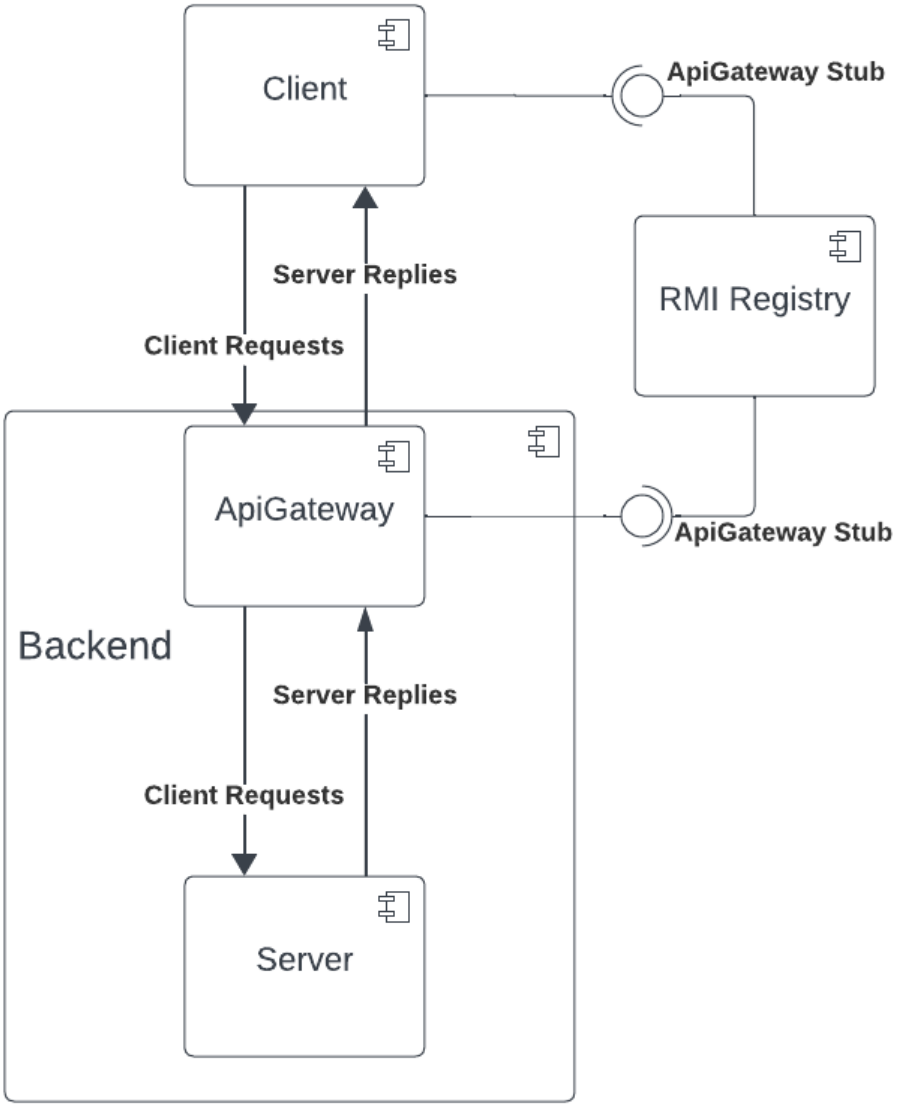
o. sell double auction

method: public String sellDoubleAuction(String email, String password, int auctionId, int sellingPrice) throws RemoteException
purpose: add a selling price to a double auction
exceptions:

- using a double auction ID that doesn't exist in the system
- same user is trying to place another buying/selling price on the same double auction
- using a negative number for the selling price
- trying to place a selling price on a double auction that has no more sellers

2. Architecture:

2.1. High-Level Architecture Diagram:



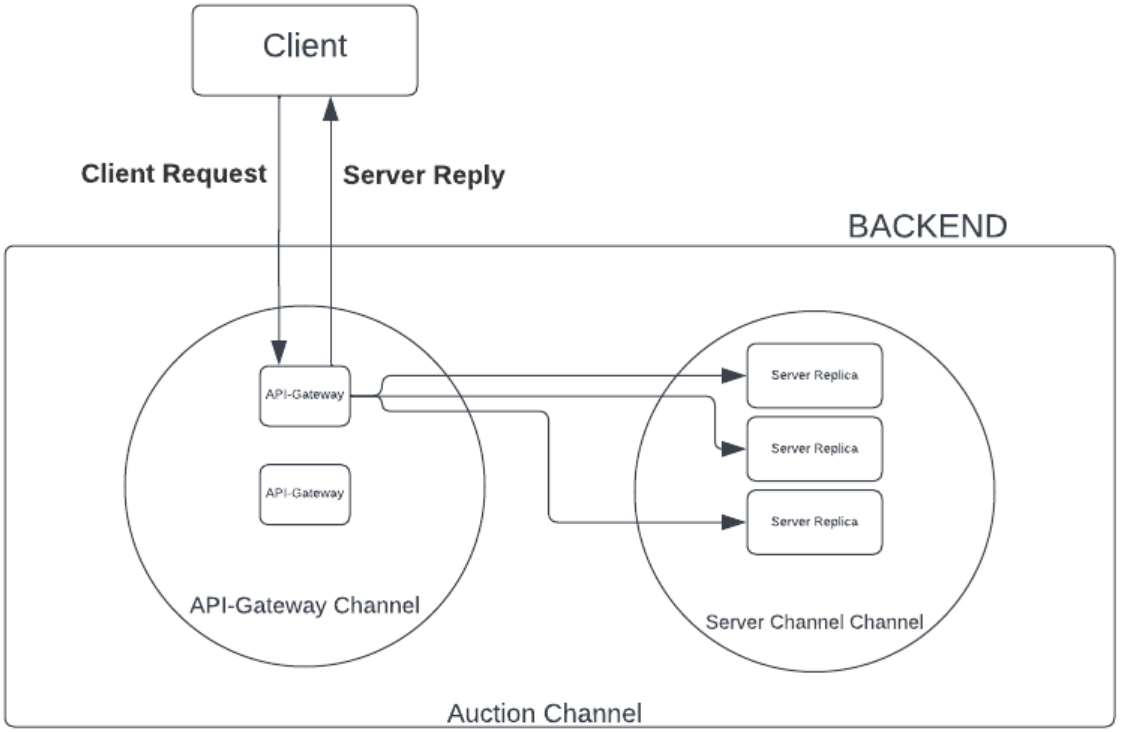
2.2. Layering & Tiering:

For the Auction System, we are using a 2-Tier Architecture

Presentation Tier: the presentation tier handles the user interface for the auction system. The client module is responsible for the presentation tier

Business-Logic & Data Tier: the business-logic and data tier handles the logic of how the auction system works and stores all the data needed by the auction system. The Server Module is responsible for the business-logic & data tier. Both tiers are at a single level because the business-logic and data reside in a single place which is the server processes, and we don't have a separate database or persistent data storage for storing the data

2.3. Backend Distributed Systems Architecture



API-Gateway Channel:

- The coordinator in the API-Gateway Channel connects to the RMI registry and receives client requests
- All other members, except the coordinator, in the API-Gateway Channel do nothing at all
- When the coordinator in the API-Gateway Channel crashes and leaves the channel a new member in the API-Gateway Channel will become the coordinator and connect to the RMI Registry and will be responsible for receiving client requests

Auction Channel:

- The Auction Channel contains all the members in the whole back-end system (i.e. all the members in the Server Channel and the API-Gateway Channel)

- When client requests are received by the API-Gateway coordinator it will send the request to all Server Replicas through the Auction Channel using the RpcDispatcher and wait until all the server replicas return a response
- If 10 seconds have passed and the API-Gateway hasn't received responses yet from the server replicas a timeout exception will be thrown
- Once the API-Gateway coordinator receives the response from all the server replicas it picks a single response from all the received responses and sends it back to the client
- This approach allows for active replication as all the server replicas process client requests, and there is no need for coordination between the server replicas as all of them will have the same exact state after processing the same client requests in all the server replicas

Server Channel:

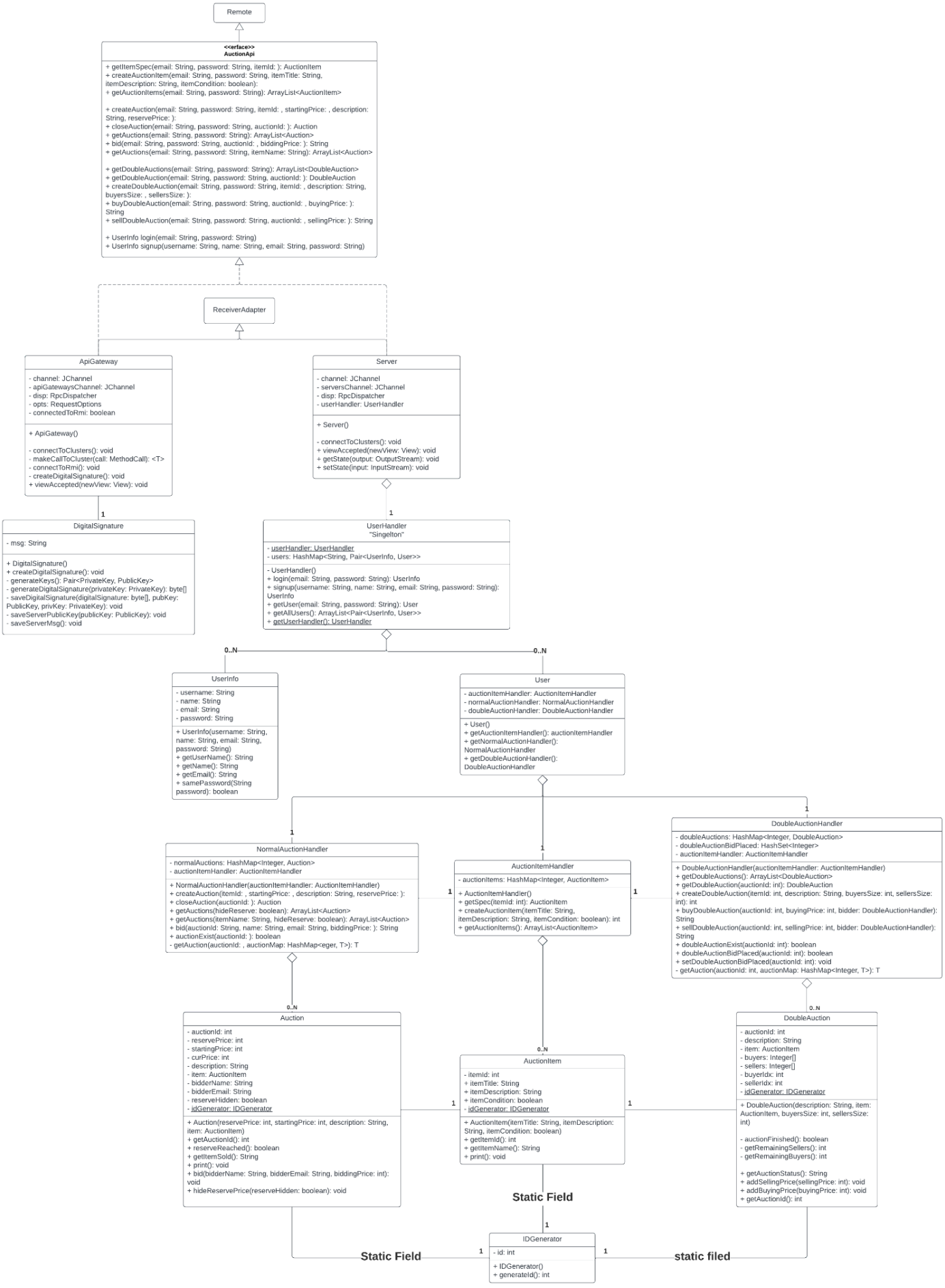
- The server channel is mainly used to allow for state transfer when new server members enter the system
- Once a new server member joins the cluster it will get the current state of the cluster from the coordinator in the Server Channel
- This implies that new server replicas can handle client requests normally and give the same exact results as all the previous server replicas

Final Remarks:

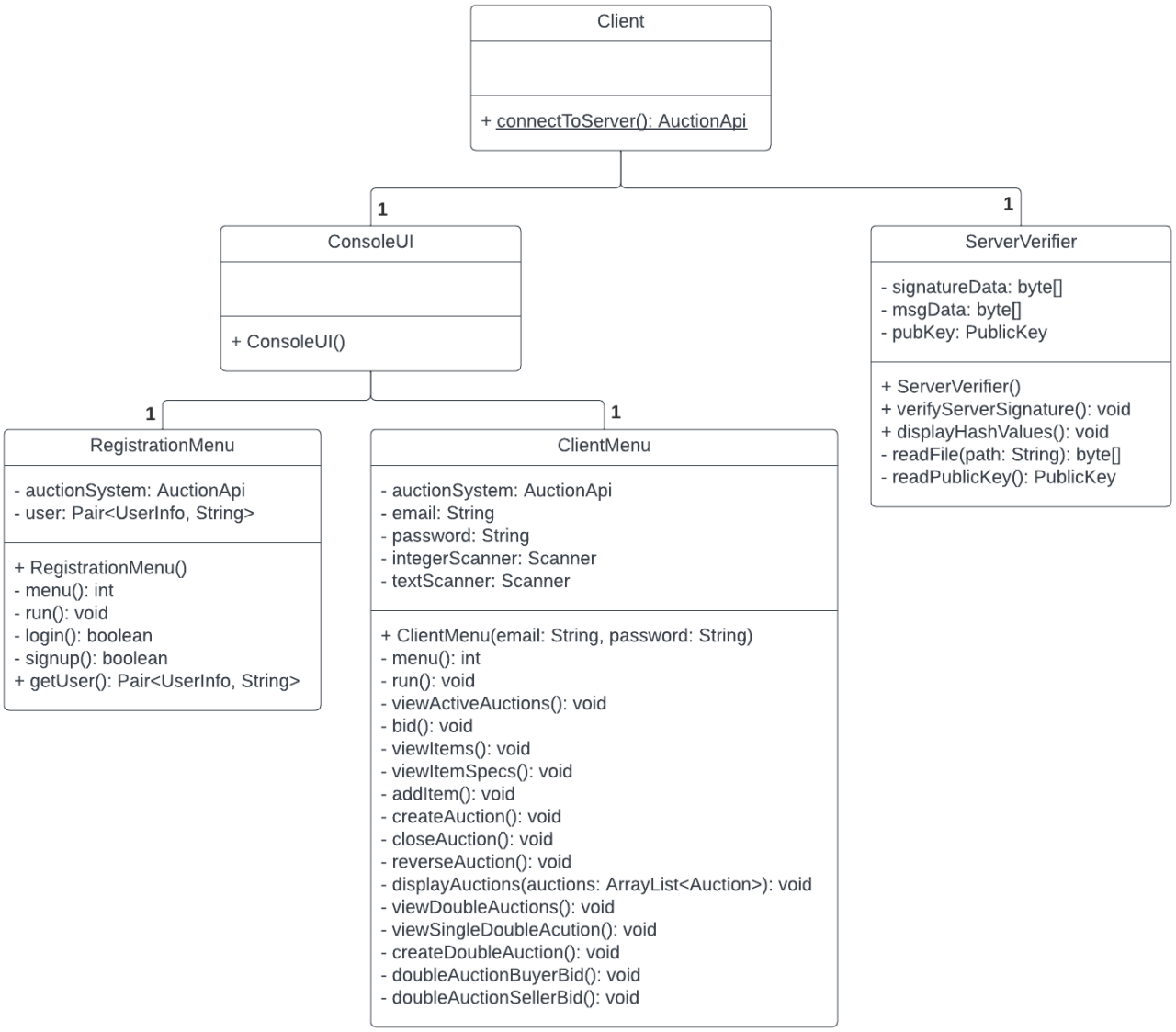
- By having Active Replication and State Transfer we can finally say that our system is Fault Tolerant
- Failures can happen in any server replica and the system will function normally due to Active Replication as other server replicas will be able to handle incoming client requests
- When new replicas join the server still will be able to handle incoming client requests due to state transfer (i.e. getting the cluster's current state)
- The only constraint is to have at least a single server replica in the cluster
- Also, when the API-Gateway coordinator fails or crashes another API-Gateway replica will connect to the RMI Registry and still be able to handle incoming client requests
- As a side note even if all the API-Gateway replicas have stopped or crashed that wouldn't be a problem given that there is at least a single server replica, because API-Gateway doesn't contain any state and when all API-Gateway replicas stop then clients will not be able to access the server and make requests but we would still have the state of the whole system and simply once a single API-Gateway replica joins the cluster the service will be back again and ready to handle client requests

3. Class Diagram:

3.1. Server Class Diagram:


Remote
<<erface>>
AuctionApi
+ getItemSpec(email: String, password: String, itemId:): AuctionItem
+ createAuctionItem(email: String, password: String, itemTitle: String, itemDescription: String, itemCondition: boolean):
+ getAuctionItems(email: String, password: String): ArrayList<AuctionItem>
+ createAuction(email: String, password: String, itemId: , startingPrice: , description: String, reservePrice:):
+ closeAuction(email: String, password: String, auctionId:): Auction
+ getAuctions(email: String, password: String): ArrayList<Auction>
+ bid(email: String, password: String, auctionId: , biddingPrice:): String
+ getAuctions(email: String, password: String, itemName: String): ArrayList<Auction>
+ getDoubleAuctions(email: String, password: String): ArrayList<DoubleAuction>
+ getDoubleAuction(email: String, password: String, auctionId:): DoubleAuction
+ createDoubleAuction(email: String, password: String, itemId: , description: String, buyersSize: , sellersSize:):
+ buyDoubleAuction(email: String, password: String, auctionId: , buyingPrice:): String
+ sellDoubleAuction(email: String, password: String, auctionId: , sellingPrice:): String
+ UserInfo login(email: String, password: String)
+ UserInfo signup(username: String, name: String, email: String, password: String)
ReceiverAdapter
ApiGateway
- channel: JChannel
- apiGatewaysChannel: JChannel
- disp: RpcDispatcher
- opts: RequestOptions
- connectedToRmi: boolean
+ ApiGateway()
- connectToClusters(): void
- makeCallToCluster(call: MethodCall): <T>
- connectToRmi(): void
- createDigitalSignature(): void
+ viewAccepted(newView: View): void
Server
- channel: JChannel
- serversChannel: JChannel
- disp: RpcDispatcher
- userHandler: UserHandler
+ Server()
- connectToClusters(): void
+ viewAccepted(newView: View): void
+ getState(output: OutputStream): void
+ setState(input: InputStream): void
DigitalSignature
- msg: String
+ DigitalSignature()
+ createDigitalSignature(): void
- generateKeys(): Pair<PrivateKey, PublicKey>
- generateDigitalSignature(privateKey: PrivateKey): byte[]
- saveDigitalSignature(digitalSignature: byte[], pubKey: PublicKey, privKey: PrivateKey): void
- saveServerPublicKey(publicKey: PublicKey): void
- saveServerMsg(): void
UserHandler
"Singelton"
- userHandler: UserHandler
- users: HashMap<String, Pair<UserInfo, User>>
- UserHandler()
+ login(email: String, password: String): UserInfo
+ signup(username: String, name: String, email: String, password: String): UserInfo
+ getUser(email: String, password: String): User
+ getAllUsers(): ArrayList<Pair<UserInfo, User>>
+ getUserHandler(): UserHandler
UserInfo
- username: String
- name: String
- email: String
- password: String
+ UserInfo(username: String, name: String, email: String, password: String)
+ getUserName(): String
+ getName(): String
+ getEmail(): String
+ samePassword(String password): boolean
User
- auctionItemHandler: AuctionItemHandler
- normalAuctionHandler: NormalAuctionHandler
- doubleAuctionHandler: DoubleAuctionHandler
+ User()
+ getAuctionItemHandler(): auctionItemHandler
+ getNormalAuctionHandler(): NormalAuctionHandler
+ getDoubleAuctionHandler(): DoubleAuctionHandler
NormalAuctionHandler
- normalAuctions: HashMap<Integer, Auction>
- auctionItemHandler: AuctionItemHandler
+ NormalAuctionHandler(auctionItemHandler: AuctionItemHandler)
+ createAuction(itemId: , startingPrice: , description: String, reservePrice:):
+ closeAuction(auctionId:): Auction
+ getAuctions(hideReserve: boolean): ArrayList<Auction>
+ getAuctions(itemName: String, hideReserve: boolean): ArrayList<Auction>
+ bid(auctionId: String, name: String, email: String, biddingPrice:): String
+ auctionExist(auctionId:): boolean
- getAuction(auctionId: , auctionMap: HashMap<eger, T>): T
AuctionItemHandler
- auctionItems: HashMap<Integer, AuctionItem>
+ AuctionItemHandler()
+ getSpec(itemId: int): AuctionItem
+ createAuctionItem(itemTitle: String, itemDescription: String, itemCondition: boolean): int
+ getAuctionItems(): ArrayList<AuctionItem>
DoubleAuctionHandler
- doubleAuctions: HashMap<Integer, DoubleAuction>
- doubleAuctionBidPlaced: HashSet<Integer>
- auctionItemHandler: AuctionItemHandler
+ DoubleAuctionHandler(auctionItemHandler: AuctionItemHandler)
+ getDoubleAuctions(): ArrayList<DoubleAuction>
+ getDoubleAuction(auctionId: int): DoubleAuction
+ createDoubleAuction(itemId: int, description: String, buyersSize: int, sellersSize: int): int
+ buyDoubleAuction(auctionId: int, buyingPrice: int, bidder: DoubleAuctionHandler): String
+ sellDoubleAuction(auctionId: int, sellingPrice: int, bidder: DoubleAuctionHandler): String
+ doubleAuctionExist(auctionId: int): boolean
+ doubleAuctionBidPlaced(auctionId: int): boolean
+ setDoubleAuctionBidPlaced(auctionId: int): void
- getAuction(auctionId: int, auctionMap: HashMap<Integer, T>): T
Auction
- auctionId: int
- reservePrice: int
- startingPrice: int
- curPrice: int
- description: String
- item: AuctionItem
- bidderName: String
- bidderEmail: String
- reserveHidden: boolean
- idGenerator: IDGenerator
+ Auction(reservePrice: int, startingPrice: int, description: String, item: AuctionItem)
+ getAuctionId(): int
+ reserveReached(): boolean
+ getItemSold(): String
+ print(): void
+ bid(bidderName: String, bidderEmail: String, biddingPrice: int): void
+ hideReservePrice(reserveHidden: boolean): void
AuctionItem
- itemId: int
+ itemTitle: String
+ itemDescription: String
+ itemCondition: boolean
- idGenerator: IDGenerator
+ AuctionItem(itemTitle: String, itemDescription: String, itemCondition: boolean)
+ getItemId(): int
+ getItemName(): String
+ print(): void
DoubleAuction
- auctionId: int
- description: String
- item: AuctionItem
- buyers: Integer[]
- sellers: Integer[]
- buyerIdx: int
- sellerIdx: int
- idGenerator: IDGenerator
+ DoubleAuction(description: String, item: AuctionItem, buyersSize: int, sellersSize: int)
- auctionFinished(): boolean
- getRemainingSellers(): int
- getRemainingBuyers(): int
+ getAuctionStatus(): String
+ addSellingPrice(sellingPrice: int): void
+ addBuyingPrice(buyingPrice: int): void
+ getAuctionId(): int
Static Field
IDGenerator
- id: int
+ IDGenerator()
+ generateId(): int
Static Field
static filed
1
0..N

3.2. Client Class Diagram:



3.3. Design Rationale (Server):

I will use a bottom-up approach to explain the rationale behind my design for the server. With a bottom-up approach, we will start from the Main Classes in the System and keep building upon them until we reach the Server Class.

- We have 3 main entities in the Auction System
    - Auction Items
    - Normal Auctions (Forward & Reverse)
    - Double Auction
- Every Auction Item, Normal Auction & Double Auction should have a unique ID and that is the responsibility of the "IDGenerator" Class
- Every Normal Auction has a single Auction Item that it will be offering
- Every Double Auction has a single Auction Item that it will be offering
- Let's now make a small jump and discuss one of the most critical design choices in the Auction System, and that is every User in the Auction System has his own Auction

Items, Normal Auctions, and Double Auctions this implies that any Auction Item, Normal Auction or Double Auction in the Program belongs to only a single User. Also, as a side note every user in our system can act as both a Seller & Buyer it would be the user's choice to act as any single one of them or even both

- Given the design decision stated at point 5 let's continue with our bottom-up analysis when stating that every User has Normal Auctions, Double Auctions, and Auction Items this implies that users can have many of each one of those which is why we have 3 classes handling each one of those independently and they are defined as follows:
    - "NormalAuctionHandler" stores all the Normal Auctions the User owns and defines all the operations that the User can perform on the Normal Auctions he owns
    - "DoubleAuctionHandler" stores all the Double Auctions the User owns and defines all the operations that the User can perform on the Double Auctions he owns
    - "AuctionItemHandler" stores all the Auction Items the User owns and defines all the operations that the User can perform on the Auction Items he owns
- When creating a Normal Auction the Normal Auction Handler will receive the item ID for the item of the Normal Auction and that is why the Normal Auction Handler needs to know what items the user has to check if the user has this item ID and to convert the item ID to a real AuctionItem Object. The Same applies to the DoubleAuction Handler. That is why both the "NormalAuctionHandler" and the "DoubleAuctionHandler" receive the "AuctionItemHandler" of the user in their constructors
- For the "User" Class it just contains an instance of the "AuctionItemHandfdler", "NormalAuctionHandler", and the "DoubleAuctionHandler", and getters for all those handlers
- Now till this point, we have discussed the main design of the Auction System. Now there are 2 main aspects we haven't discussed yet the 1st is that our system allows users to Login & Signup, 2nd is that we can have a lot of users in the Auction System
- The "UserInfo" Class stores all the user credentials like the email, password, etc.
- Now let's move to one of the most important classes: "UserHandler" The user handler does 2 main things: stores all users in the System, implements the logic of login and signup functionalities, and enforces user authentication during login and signup. Something to note is that the "UserHandler" Class is a Singleton which means there can only exist one instance of the "UserHandler" Class
- So after reaching this point, we have already discussed all the main design choices of the system and the rationale behind it
- Let's conclude our discussion with the following additional points about the design:
    - The "Server" class has a single instance of the "UserHandler" Class
    - The "Server" Class implements the "AuctionApi" Interface which is the interface the client expects to use to make requests
    - The "ApiGateway" Class also implements the "AuctionApi" Interface as the API-Gateway is responsible for receiving client requests
    - The "ApiGateway" Class has a single instance of the "DigitalSignature" Class and the API-Gateway is responsible for creating digital signatures for clients to validate the server identity, and simply we create Digital Signatures in the API-Gateway as the Client Communicates only with the API-Gateway and the API-Gateway is the class which connects to the RMI Registry

4. How to run the code:

Server Side:

compile: javac -cp jgroups-3.6.20.Final.jar;. *.java
RMI Registry: rmiregistry
Running Api-Gateway: java -cp jgroups-3.6.20.Final.jar;. ApiGateway
Running Server: java -cp jgroups-3.6.20.Final.jar;. Server

Client Side:

compile: javac *.java
Running Client: java Client