In our first Group meeting:

We choose an appropriate project and refine the requirements

The project is as follows and the requirements are all architecturally significant.

Going...Going...Gone!

An auction company wants to take their auctions online to a nationwide scale--customers choose the auction to participate in, wait until the auction begins, then bid as if they were there in the room, with the auctioneer.

- Users: scale up to hundreds of participants (per auction), potentially up to thousands of participants, and as many simultaneous auctions as possible
- Requirements:
  - auctions must be categorized and 'discoverable'
  - auctions must be as real-time as possible
  - auctions must be mixed live and online
  - video stream of the action after the fact
  - handle money exchange
  - participants must be tracked via a reputation index
- Additional Context:
  - auction company is expanding aggressively by merging with smaller competitors
  - if nationwide auction is a success, replicate the model overseas
  - budget is not constrained--this is a strategic direction
  - company just exited a lawsuit where they settled a suit alleging fraud

(Cited from http://nealford.com/katas/list.html).

Refinement of the above requirements:

The refined requirements above have been marked with red color

- Store the auction video in a database.
- Every user has a reputation record, some behavior can favor the reputation, while some can impair it.
- Try to build this product basing in some common OS, framework and protocol.
- Build a customer service system such that the company will have more chance to communicate with the customer.

We try to imagine how our product work if we successfully complete the design work. With the concept of Object-oriented programming, we design the basic structure of the product.

Imagine this scenario:

The users open our client-side, search the auction information, participant in their interested auctions. When they enter into one auction, they can watch the real-time video and bid. And when they succeed in some biding, they will need to pay for the items. During this process, the client-side will send many requests to the servers, and servers will process these requests and return the result to the client-side. Meanwhile, the servers keep receiving the data flow from the auction scene and response to some requests from auction scene.

According to object-oriented programming concept, we design the basic architecture (Figure 1, see it in appendix).

Consider the functionality of the three components respectively, the relations between auction scene and servers, and the relations between servers and client-side,

Requirements:

auctions must be categorized and 'discoverable'**(search in client-side).**

auctions must be as real-time as possible **(video stream: auction scene- servers- client_side).**

auctions must be mixed live and online **(auction scene and real-time video in client_side).**

video stream of the action after the fact **(Database).**

handle money exchange **(payment system).**

participants must be tracked via a reputation index **(reputation system).**

company just exited a lawsuit where they settled a suit alleging fraud **(customer service system).**

we made further detailed design: Figure 2 (see it in appendix).

We divided the design work into five pieces and distributed these pieces to five team members evenly. Here are the division the design work:

Qi Zhao: Auction scene.
Yuandong Chen: Login/register system, bid system, reputation system, payment system.
Kebo Duan: Video system, Customer service system.
Zhuo Zheng: Search system, Database.
Yunlong Zhao: Client-side.

In the end of this meeting, we talked about how these different components are cooperated to implement some specific functionality of this product. And we summarized what design decisions and tradeoff we have made.

Design decisions:

(1) As the Figure 2 shows, we have divided the overall system into many sub-systems.
(2) We would develop multiple versions of the client-side basing on some common operating systems (Android, Mac OS, Windows).

(3) We would adopt self-build servers.

Tradeoff:

We need to choose between cloud servers and self-build servers.

	Cost	Reliability	security	agility
Cloud servers	first lower, later higher	higher	Low data security	high
Self-build servers	First higher, later lower	lower	High data security	low

According to these requirements:

- Users: scale up to hundreds of participants (per auction), potentially up to thousands of participants, and as many simultaneous auctions as possible
- auction company is expanding aggressively by merging with smaller competitors
- budget is not constrained--this is a strategic direction

we will choose the self-build servers.

In our second group meeting:

1. Every team member showed the resulting architectures of his pieces of work. And explained to others how he designed the resulting architecture, what requirements he derived, and what tradeoff he has made.

Qi Zhao:

Auction Scene Module

The auction scene is divided into two parts. The first one is Video system which is used to video the auction, process the video data and stream the video packets to server layers. The second one is device control systems which is used to follow the instructions coming from the server layers, and control the display of the screen and the broadcast of the loudspeaker.

The static architecture and dynamic architecture of auction scene are shown in figure 3, figure 4(see it in appendix) respectively.

Derived Requirements：

Real-time: video delay no more than X seconds

Sharpness: about Y pixels

Sound distortion: Z degree

Trade study:

Criteria	Mandatory	Weight	Scale
Video delay no more than X seconds	Yes	100%	Seconds
Video Sharpness no less than Y	Yes	100%	Pixel
Distortion(video signal and audio signal) no more than Z	Yes	100%	Degree

According to the above trade study table, the developers would choose appropriate camera, loudspeaker, screen in the auction scene, and transport protocol.

Yuandong Chen:

Auction System/ Reputation System

Requirements:

- Auction must be as real-time as possible: every room is connected with every clients' front end devices by protocol based on TCP.
- Participants must be tracked via a reputation index: We will increase client's reputation if he successfully pay the order and vice versa. Also, clients need to provide their security id for registration. If a client's reputation (credibility) is lower than certain value that a room is required, s/he is not allowed to enter the room.

Whenever there is an auction, administer will configure the parameter the auction and send all the information to auction server, the server will spawn an actor to handle all the things happening to this room. The actor is the representation of room in our design. For example, client could send by message to the room, the room will check if it is the earliest and highest request and if it is, it will update corresponding table in the database. The reputation module will update database according to the auction result.

Basically, the auction room is a finite state machine. It can only handle certain events in certain states, not like server which is able to handle all the messages coming from everywhere.

Please see the static architecture design in figure 5 (see it in Appendix).

Login/Register/Authentication System:

When clients provide email and password to login, the login system will decode the password and match it to the corresponding email to verify the login action.

Clients must provide information about name, email, phone and password to register. Register system will save all the information into database.

Authentication system must use the restraints to verify clients' input before querying database, like password must be longer than 6 characters.

Please see the static architecture in figure 6(see it in Appendix).

Payment System:

Requirements:

- Handle money exchange: Payment system will do the exchange and ask for any kind of money the clients like. However, in auction room, every order is calculated in dollar.

Payment server will fetch order in our database. If there is any unpaid order, it will ask our email/phone server to remind client to pay the money,and spawn an actor to handle the payment transaction. The actor is connected with third party payment platform like Venmo, PayPal etc. After client paid the order, it will increase client's credibility and update order saved in the database.

Please see the static architecture in figure 7(see it in Appendix).

Kebo Duan:

Video System

derived requirements

1) Send video recorded by cameras at auction rooms to user apps.
2) Record video and store it in the storage system and can retrieve it at any time.
3) Can support hundreds of auctions simultaneously.
4) Can push video streams to of thousands of users and can potentially support millions of users in the future.
5) Support user authentication and authorization.
6) Can reconnect to the onsite DVR systems when losing the connections.
7) Can reconnect to the user apps when losing the connections.

Please see the static architecture and dynamic architecture in Figure 8 and Figure 9 respectively.

Architecture

1) User App and Living Media Server communicate through publisher/subscriber mechanism. To participate an auction, user apps register/subscribe the corresponding

auctions topic. The Living Media Server publish video streams to the auction topic according to the auction room number.

2) Living Media Server slices the living streams into small pieces and save them to a series of small video files. It supports writing files to local disks and to remote storage systems such as NAS (Network-attached storage) system.

3) Living Media Server accesses the database through an adapter, which hides the implementation details of specific database. Therefore, it does not need to change anything if vendor of the database is changed.

4) Living Media Server can be scaled out easily. It works in cluster mode and every node can provide streaming services. New node can be added when load of current node is too high.

Customer Service System

Derived requirements:

1) Support use connections through phone (voice).
2) Support use connections through internet (text).
3) Support user authentication and authorization.
4) Can provide status of transactions to customers
5) Can update status of customers' transactions.
6) Can show recorded video to customers.

Please see the static and dynamic architecture designs in Figure 10 and Figure 11(Appendix 1) respectively.

Architecture

1) Liking the Living Media Server system, the Custom Service System also accesses the database through an adapter, and therefore, it does not need to change anything if vendor of the database is changed.

2) It includes a voice recognition system that can serve customers automatically.

3) It contains a NLP (Natural Language Processing) module that can chat will customers and answer their questions.

Zhuo Zheng:

Search System

Requirements:

- Auction must be categorized and discoverable.

When users send a search query with keywords, the search system will compare the keywords with the category field in the item table in DB, then DB gives back the corresponding items' room id to the search system. According to the room id, the search system assembles room object and give it to clients.

Please see the static architecture design in Figure 12 (Appendix).

Database

Requirements:

- Participants must be tracked via a reputation index.
- Database should response to requests from various system in the server layers(register/login system, auction system, payment system, reputation system, video system, search system).

Our database has five tables: user table, item table, room table, room_user table and order table. User table is used in login/register/Auth system and bidding system. Item table stores the

information of auction items like they are in which room and the current price. A room represents an auction, this table records the information of this auction. Room_user table is specifically used for checking whether the user is allowed to bid. The last table order stores which user gets the item.

The table are attached in the Appendix 2.

Yunlong Zhao

Client-side:

Requirements:

- auctions must be categorized and 'discoverable'(search)
- auctions must be as real-time as possible (real-time video)
- participants must be tracked via a reputation index (personal information)
- auction company is expanding aggressively by merging with smaller competitors (base on common Operating systems).
- if nationwide auction is a success, replicate the model overseas (oversea-version).
- company just exited a lawsuit where they settled a suit alleging fraud (customer service).

The client-side has three parts: account module, room module and homepage. Account module is used to login in, display history order information, display personal information, pay the item, seek help, set configuration for the application and read the legal and privacy notices. Room module is used to play current video and bid. In the homepage, you can search room number, search merchandise information and display current bidding room.

If the nationwide auction is a success, an oversea-version of the client-side will be developed. We will have Chinese-version, Japanese-version, Spanish-version and so on.

Please see the static architecture design in Figure 13(Appendix 1).

After showing each one's work, we discussed how to simulate a dynamic process of the auction.

Auction System: the auction system is basically a long-running server, which accepts all kinds of events, like add-room, delete-room, reserve-room etc. When it is asked to add a room, it will spawn an actor (lightweight process) to handle the auction in a room, so the actor is the virtual representation of our room. This actor is a finite state machine, which only responds to certain events at certain states.

In auction start state, administer is free to modify the parameters like items' initial price, limit number of clients allowed to enter this room, expiration time and so on. Then the administer sends "start event" to the actor, it will commit action1 and transfer to "auction processing" state. In auction processing state, clients might send "ask" event. In checking next item state, the actor will check whether there is another item to be auction. It will go back to "auction processing" state and fetch the next item in the auction list set by the administer, otherwise, the actor will go to the "end of auction" state. Clients are allowed to enter or leave the actor in any states between the start and the end.

One actor for one auction is practical since there are at most thousands of clients in the same room at most.

Please see the dynamic architecture of the auction system in Figure 14.

Then we derive some other requirements about the framework， protocol, and programming languages.

Requirements：

Job Market: Find competent programmers.

Performance: Overhead of language itself and performance of library

Productivity: Ease of programming, ability to quick prototype

Here is the table of trade study

Criteria	Mandatory	Weight	Scale
Job Market	No	40%	3 for best Interoperability 1 for worst interoperability
Productivity	No	50%	3 for best productivity 1 for worst productivity
Performance	No	10%	3 for fastest 1 for slowest

We will choose appropriate framework, protocol, and programming languages according to the trade study table.

In the end, we summarize the work we have completed

The static architectures for every components are completed. We also simulated a dynamic process of the auction system and derived some other requirements.

In our third group meeting:

We change from self-build servers to cloud servers at the first. If our clients number are growing up to be a very significant figure , we will transfer from cloud servers to self-build servers.

The reason are as follows:

(1) We are an auction company and we have never tried this online business, cloud server are much easier and cheaper for us to use. Besides, the reliability of cloud server is higher than self-build servers.

(2) We may transfer from cloud server to self-build servers later. So, we will choose some protocol, framework, environment and programming language that are easy to transfer from cloud server to self-build server to build our system.

We discussed how to deploy these systems.

Please see the static architecture design in Figure 15Appendix).

Appendix:

Figure 1

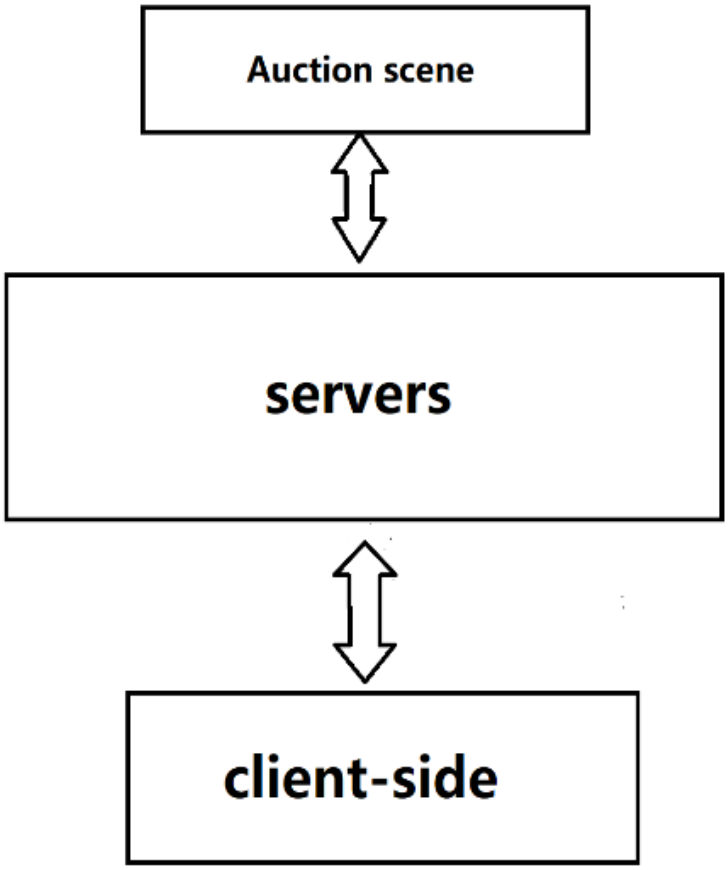


Figure 2:

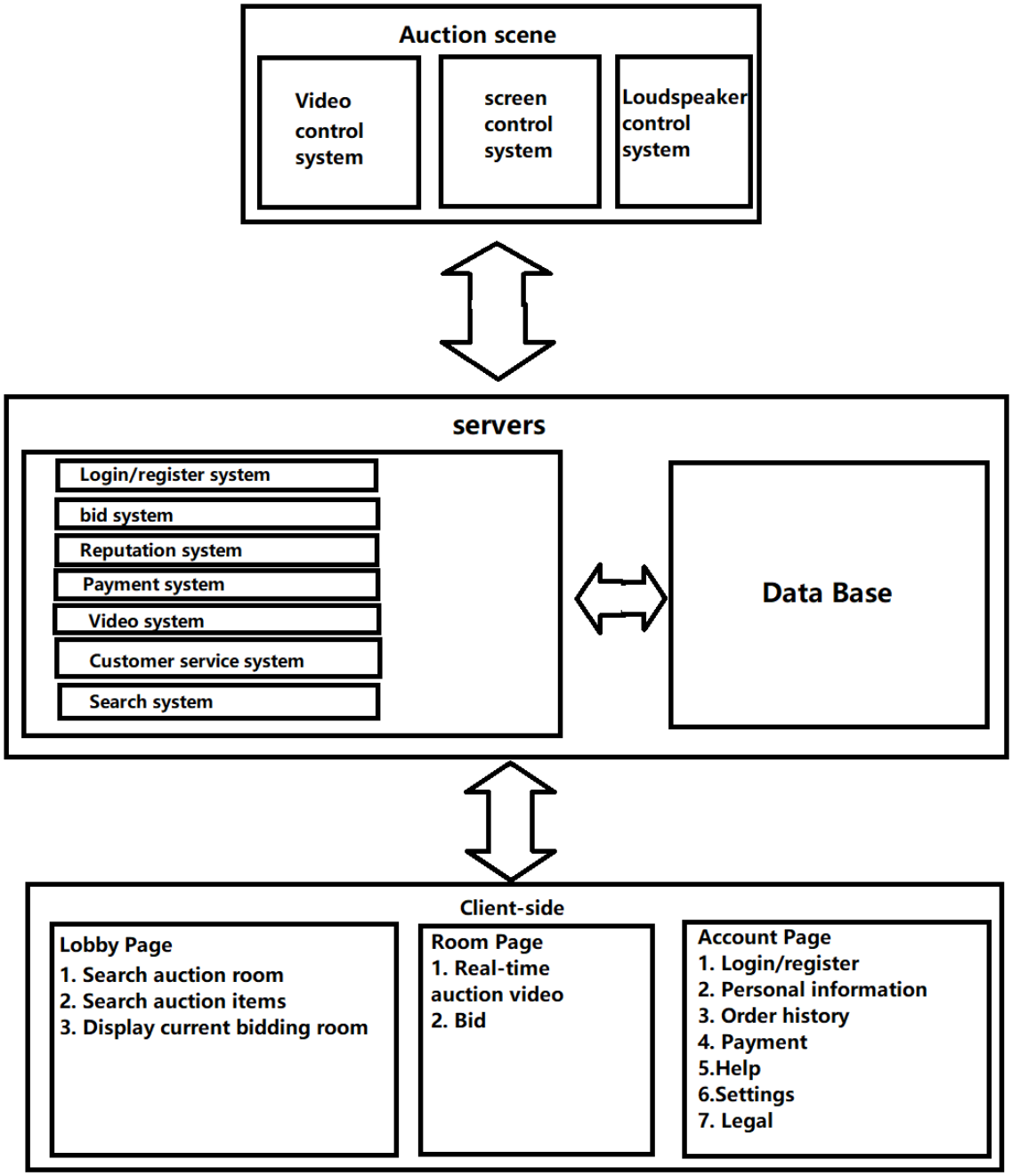

Auction scene
Video control system
screen control system
Loudspeaker control system
servers
Login/register system
bid system
Reputation system
Payment system
Video system
Customer service system
Search system
Data Base
Client-side
Lobby Page
1. Search auction room
2. Search auction items
3. Display current bidding room
Room Page
1. Real-time auction video
2. Bid
Account Page
1. Login/register
2. Personal information
3. Order history
4. Payment
5.Help
6.Settings
7. Legal

Figure 3:

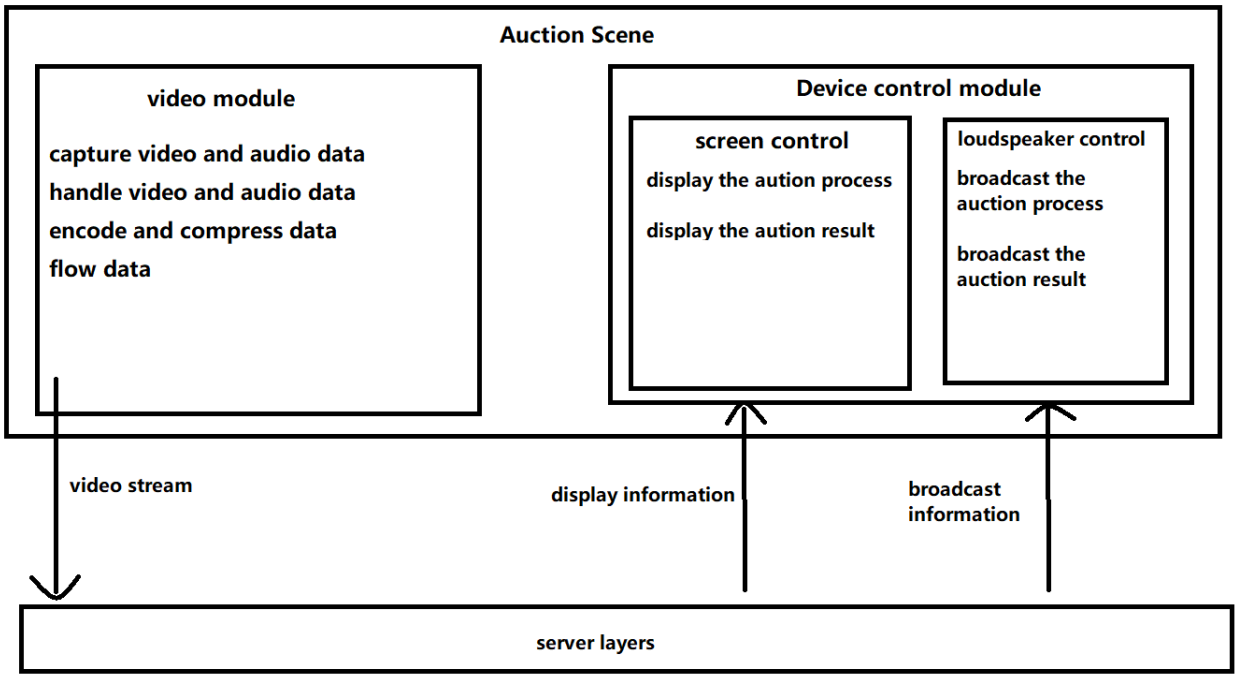

Auction Scene
video module
capture video and audio data
handle video and audio data
encode and compress data
flow data
Device control module
screen control
display the aution process
display the aution result
loudspeaker control
broadcast the auction process
broadcast the auction result
video stream
display information
broadcast information
server layers

Figure 4:

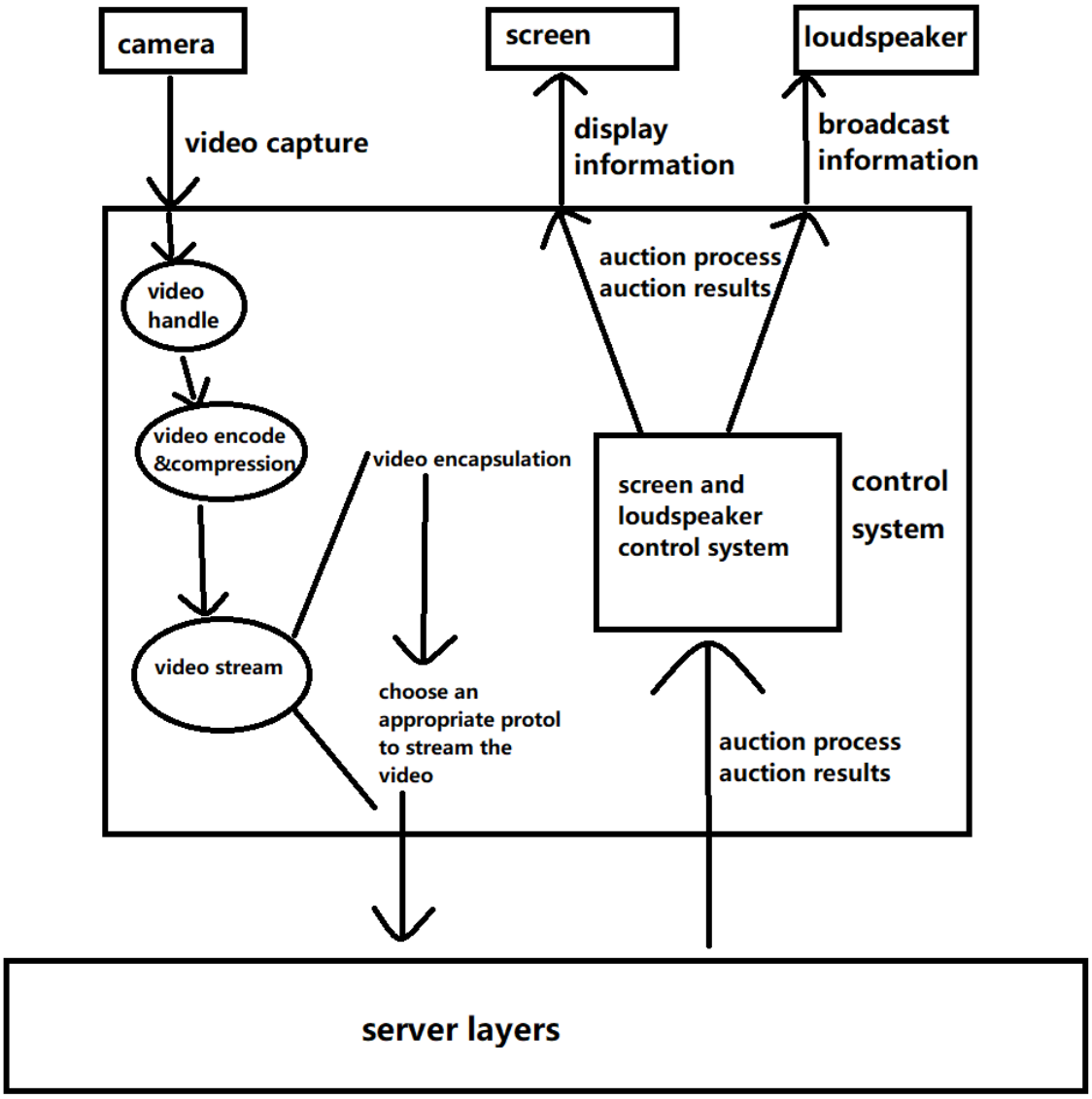

camera
screen
loudspeaker
video capture
display information
broadcast information
auction process
auction results
video handle
video encode &compression
video encapsulation
screen and loudspeaker control system
control system
video stream
choose an appropriate protol to stream the video
auction process
auction results
server layers

Figure 5:

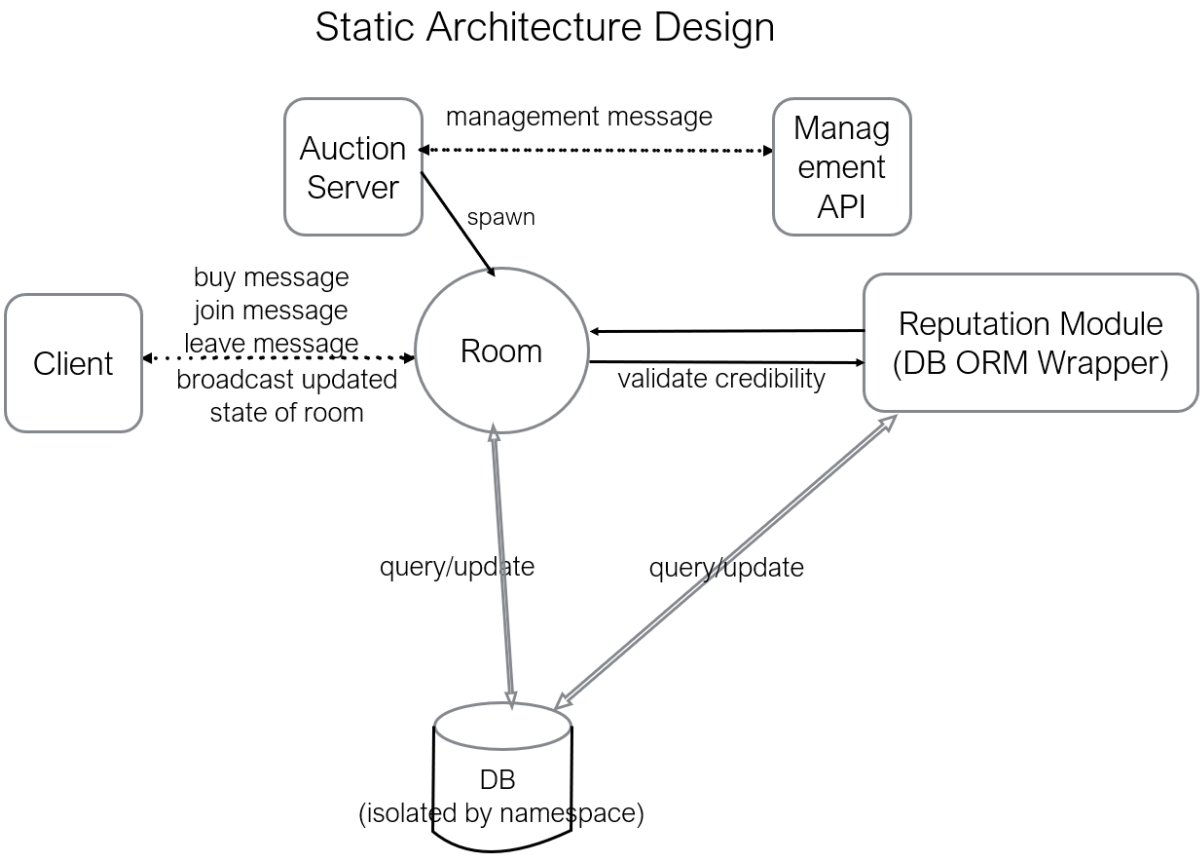

Static Architecture Design
Auction Server
management message
Management API
spawn
buy message
join message
leave message
broadcast updated state of room
Client
Room
Reputation Module (DB ORM Wrapper)
validate credibility
query/update
query/update
DB
(isolated by namespace)

Figure 6:

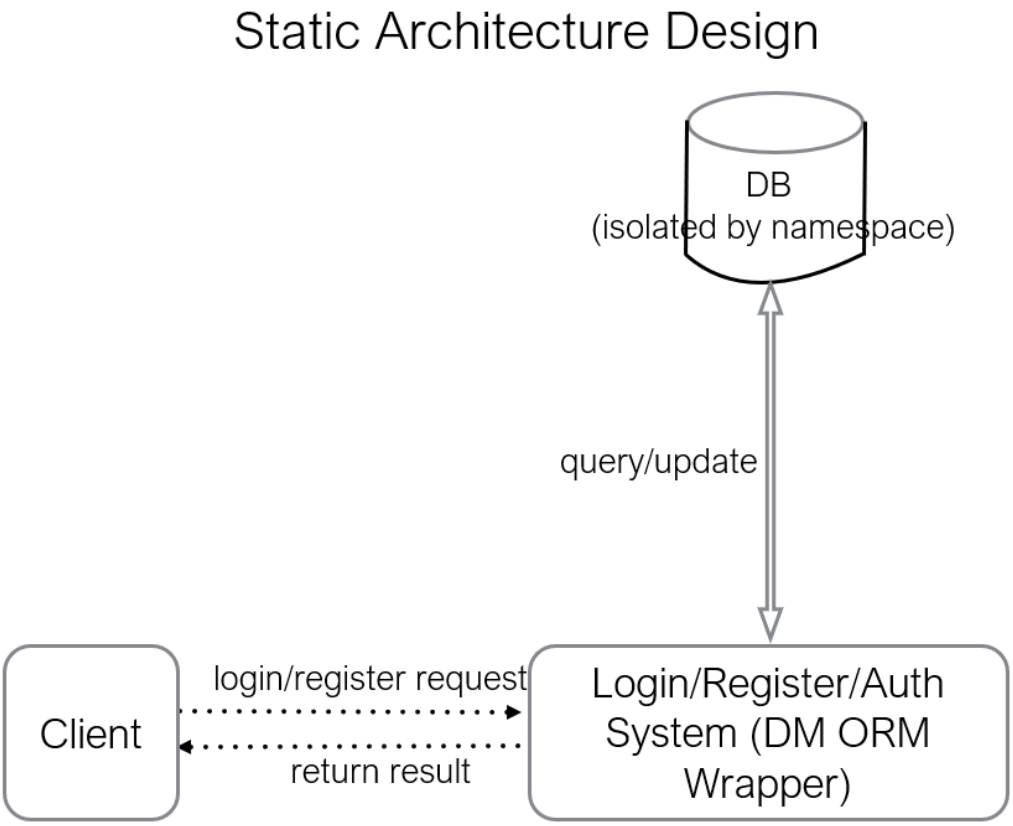

Static Architecture Design
DB
(isolated by namespace)
query/update
Client
login/register request
return result
Login/Register/Auth System (DM ORM Wrapper)

Figure 7:

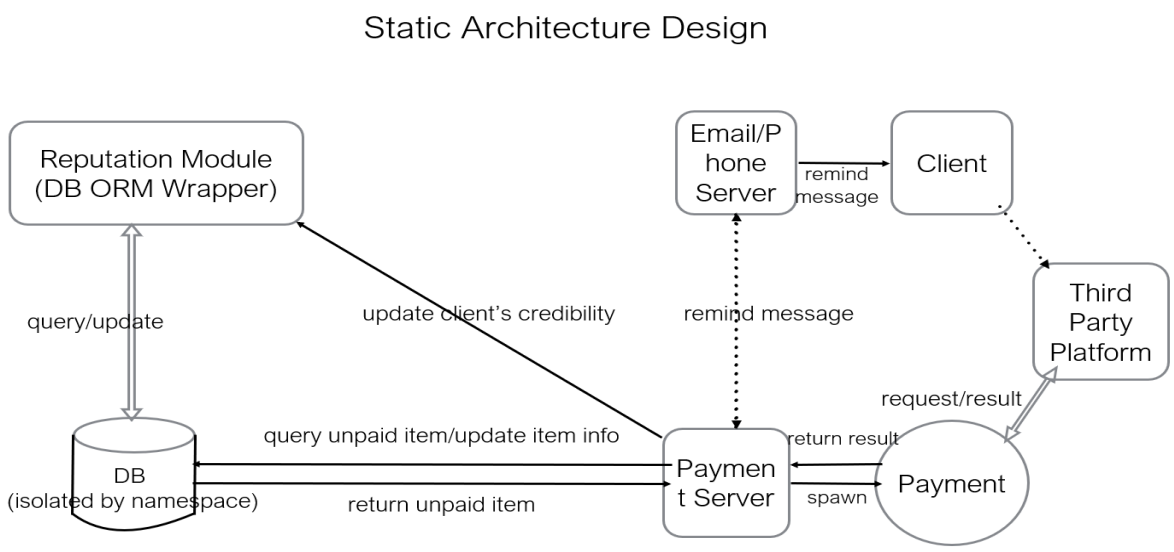

Static Architecture Design
Reputation Module
(DB ORM Wrapper)
Email/P
hone
Server
remind
message
Client
Third
Party
Platform
query/update
update client's credibility
remind message
request/result
query unpaid item/update item info
return result
DB
(isolated by namespace)
return unpaid item
Paymen
t Server
spawn
Payment

Figure 8:

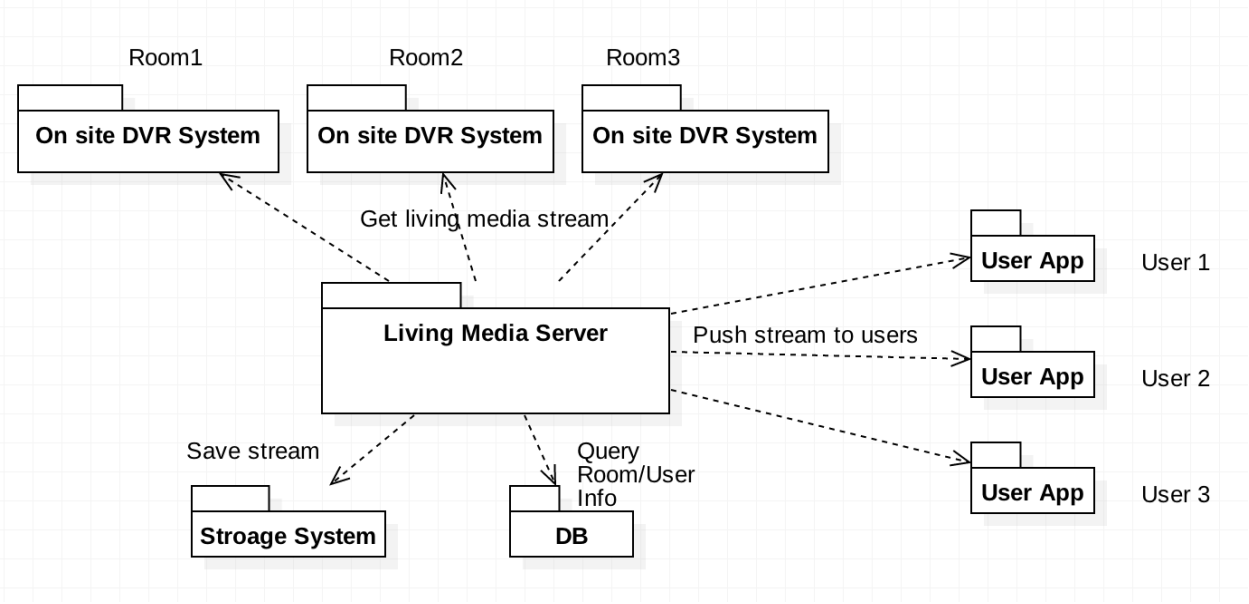

Room1
Room2
Room3
On site DVR System
On site DVR System
On site DVR System
Get living media stream
Living Media Server
User App
User 1
Push stream to users
User App
User 2
User App
User 3
Save stream
Query
Room/User
Info
Stroage System
DB

Figure 9:

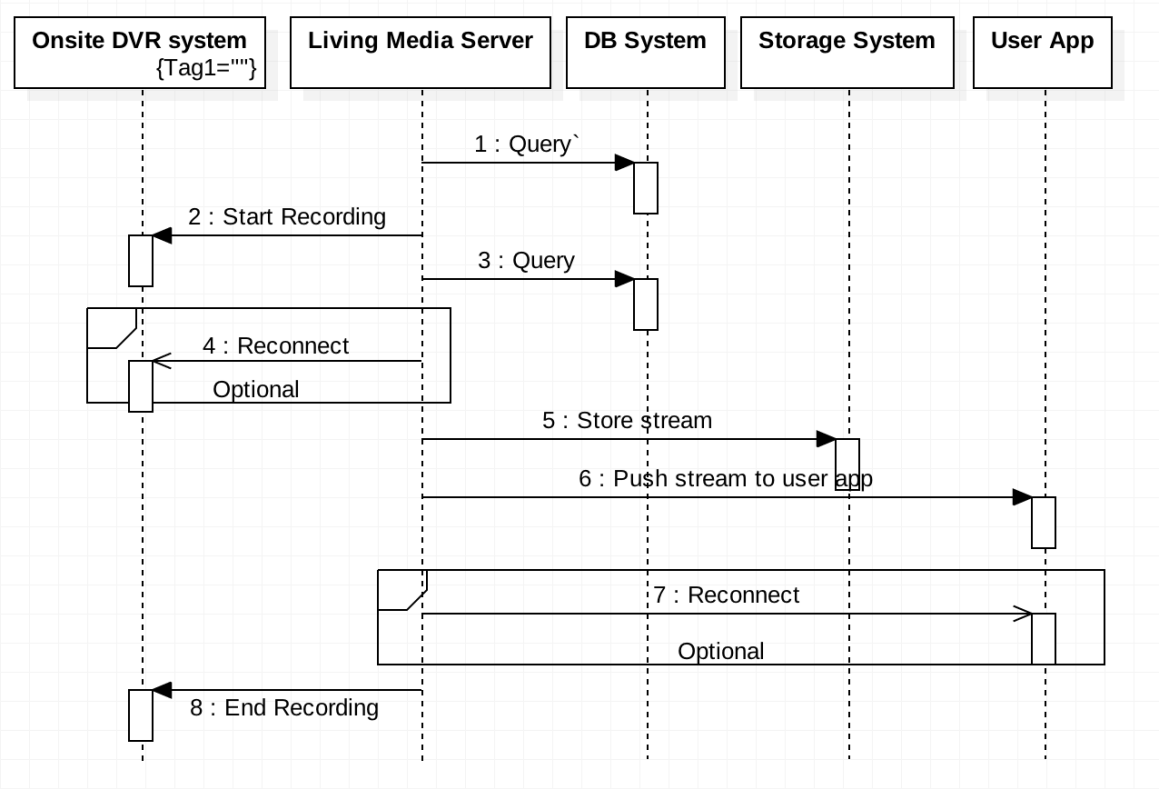

Onsite DVR system
{Tag1=""}
Living Media Server
DB System
Storage System
User App
1 : Query`
2 : Start Recording
3 : Query
4 : Reconnect
Optional
5 : Store stream
6 : Push stream to user app
7 : Reconnect
Optional
8 : End Recording

Figure 10:

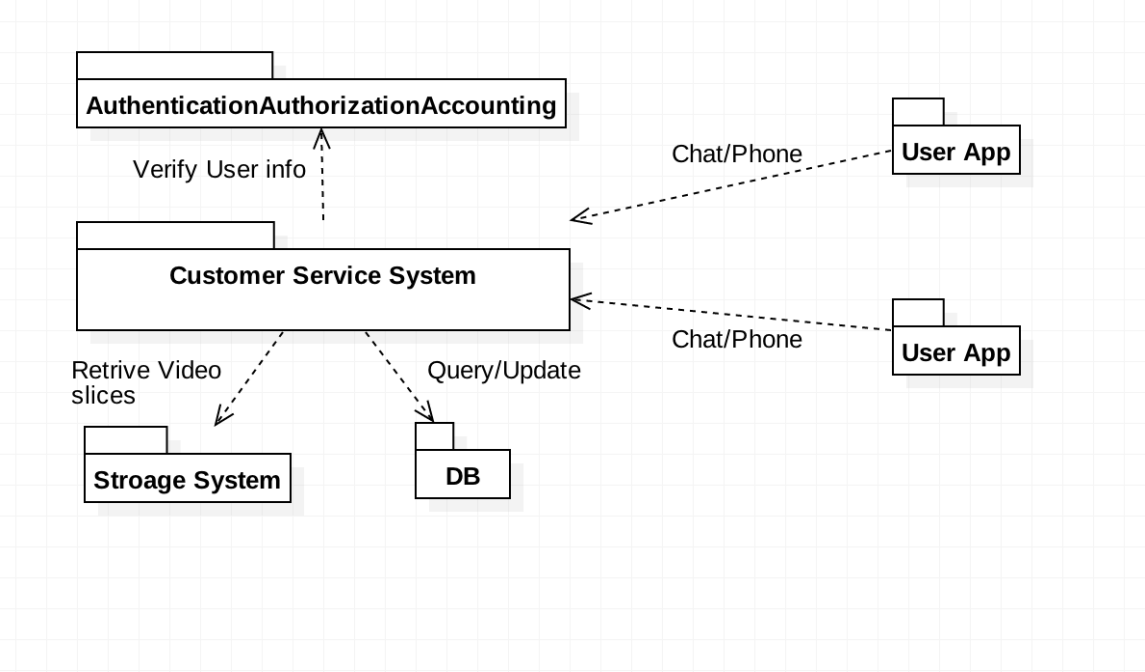

AuthenticationAuthorizationAccounting
Verify User info
Chat/Phone
User App
Customer Service System
Chat/Phone
User App
Retrive Video slices
Query/Update
Stroage System
DB

Figure 11:

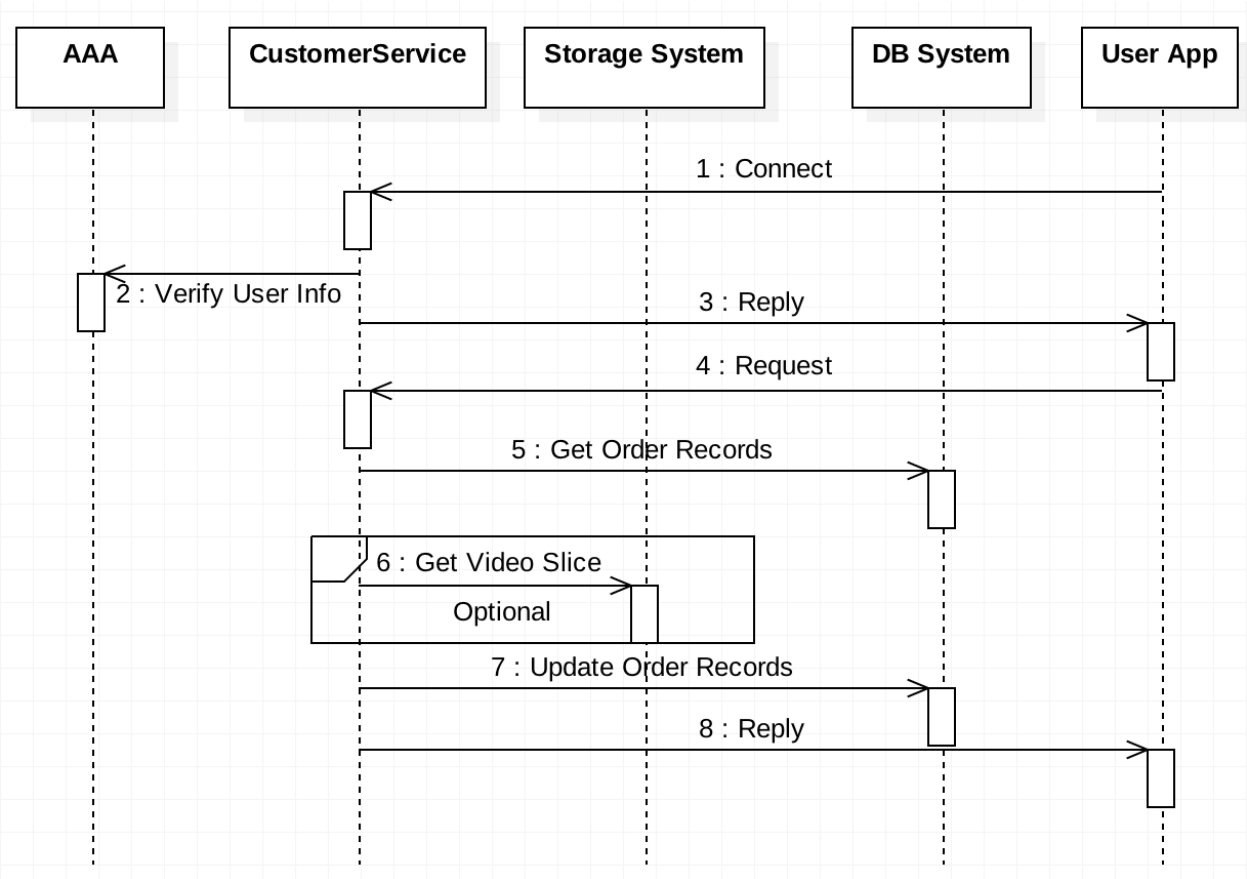

AAA
CustomerService
Storage System
DB System
User App
1 : Connect
2 : Verify User Info
3 : Reply
4 : Request
5 : Get Order Records
6 : Get Video Slice
Optional
7 : Update Order Records
8 : Reply

Figure 12:

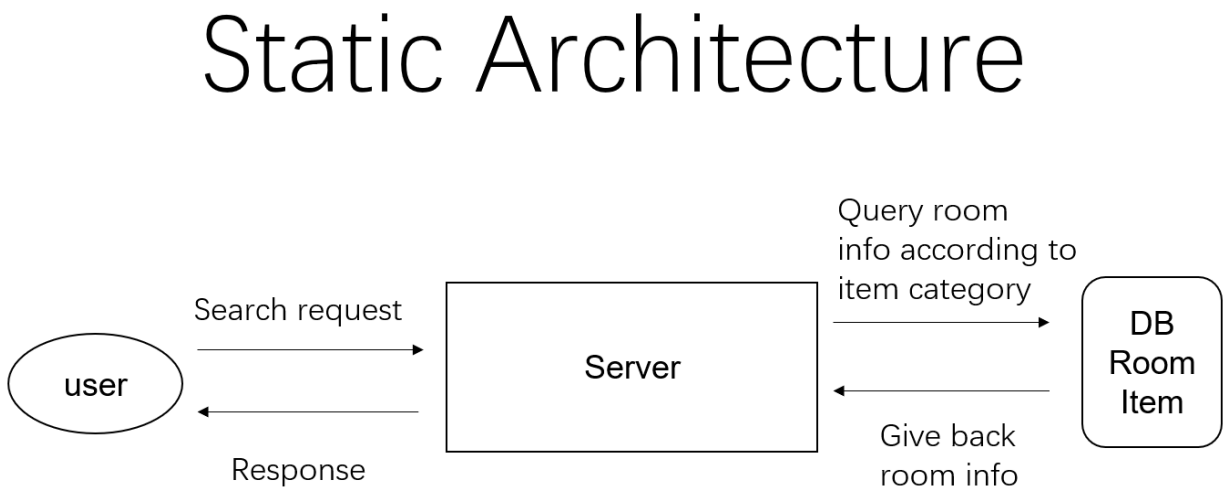

Static Architecture
Query room info according to item category
Search request
user
Server
DB
Room
Item
Response
Give back room info

Figure 13:

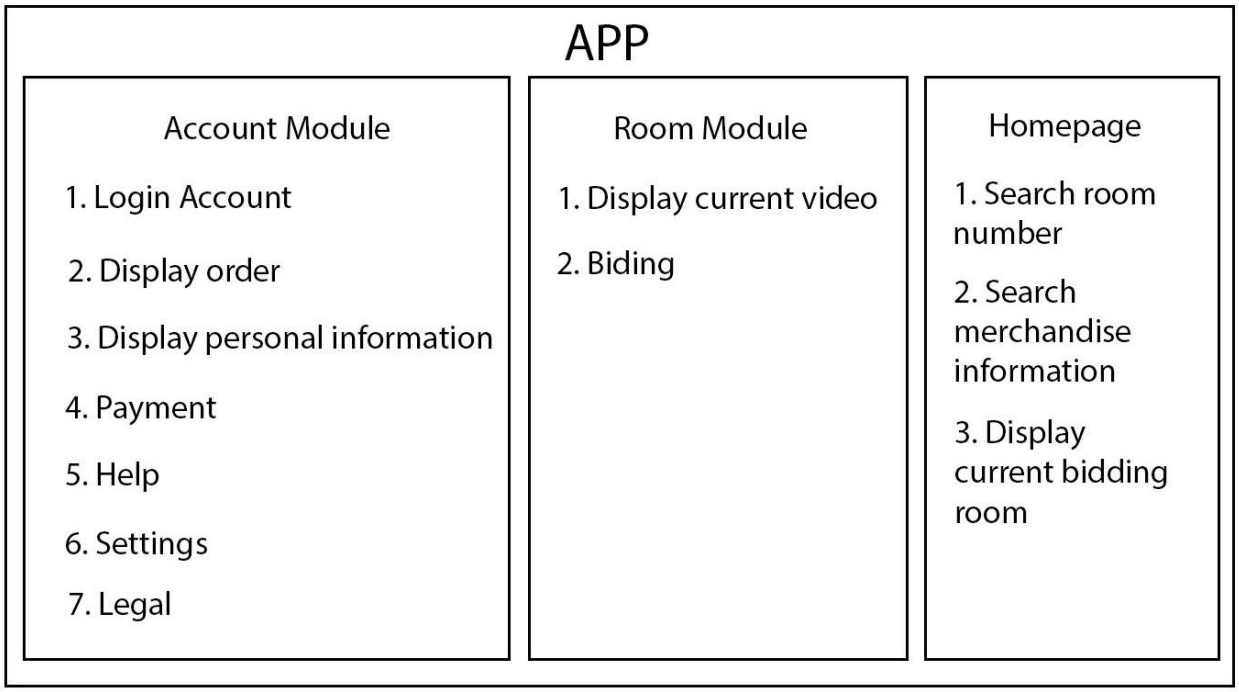

APP
Account Module
1. Login Account
2. Display order
3. Display personal information
4. Payment
5. Help
6. Settings
7. Legal
Room Module
1. Display current video
2. Biding
Homepage
1. Search room number
2. Search merchandise information
3. Display current bidding room

Figure 14:

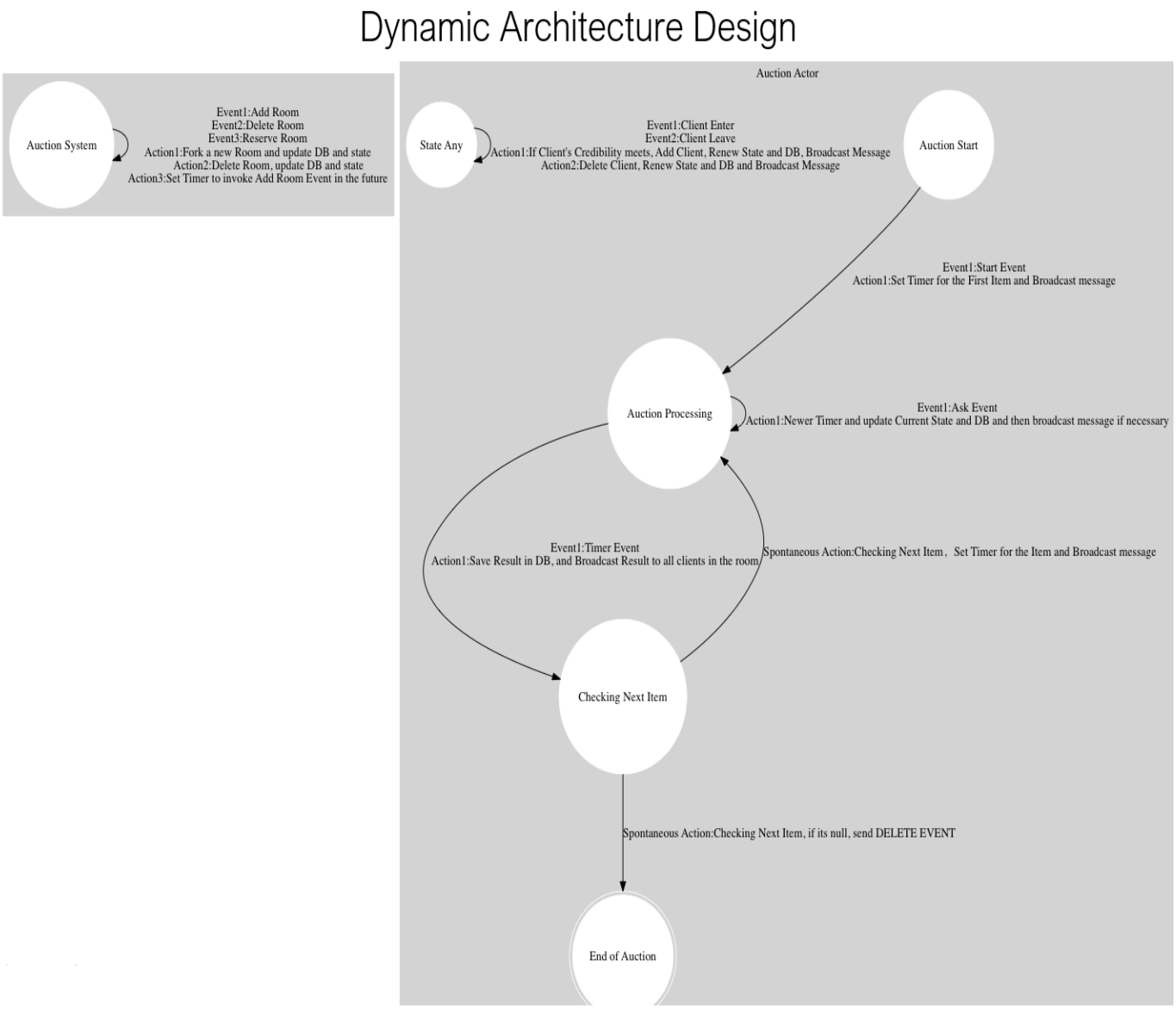
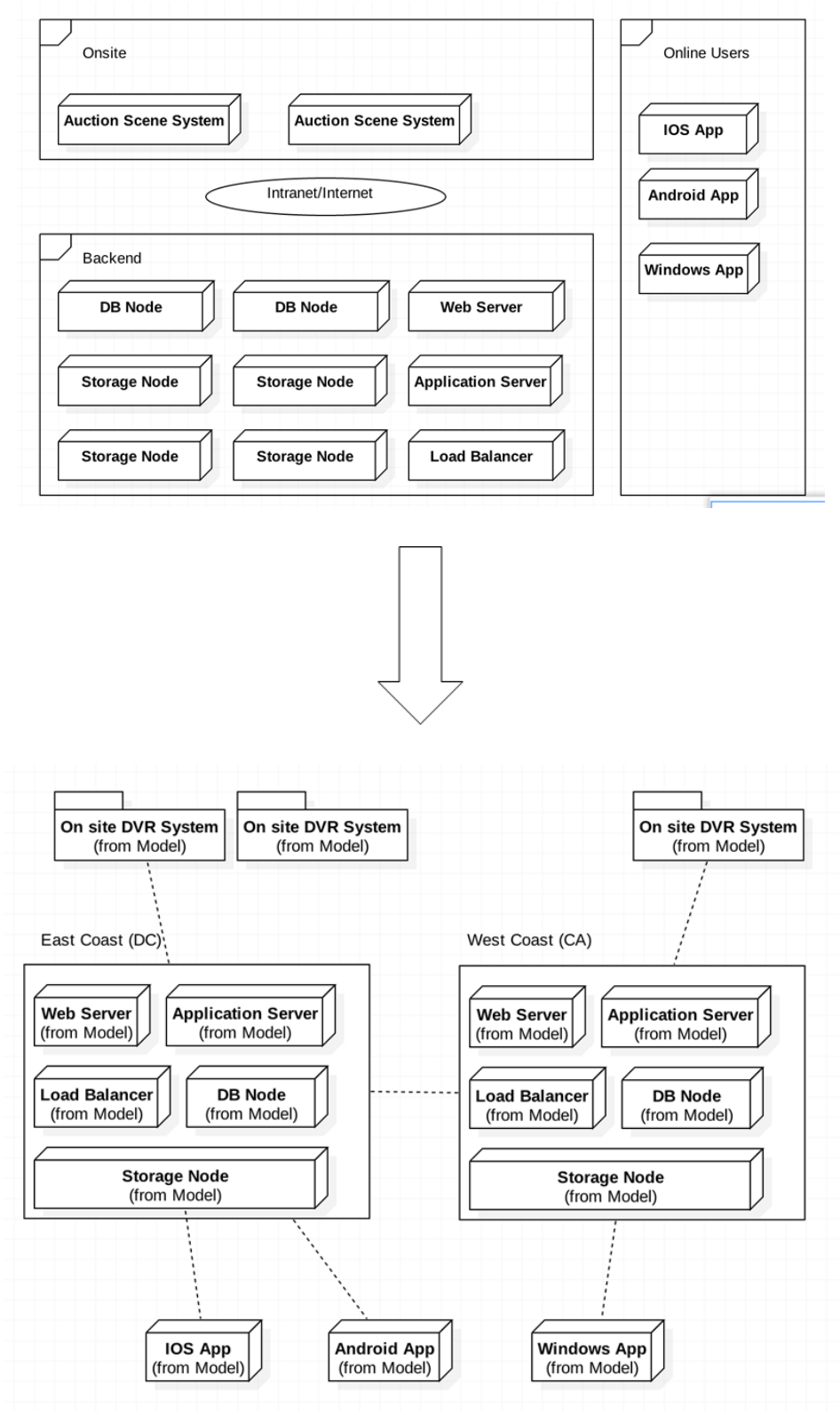

Dynamic Architecture Design
Auction System
Event1:Add Room
Event2:Delete Room
Event3:Reserve Room
Action1:Fork a new Room and update DB and state
Action2:Delete Room, update DB and state
Action3:Set Timer to invoke Add Room Event in the future
Auction Actor
State Any
Event1:Client Enter
Event2:Client Leave
Action1:If Client's Credibility meets, Add Client, Renew State and DB, Broadcast Message
Action2:Delete Client, Renew State and DB and Broadcast Message
Auction Start
Event1:Start Event
Action1:Set Timer for the First Item and Broadcast message
Auction Processing
Event1:Ask Event
Action1:Newer Timer and update Current State and DB and then broadcast message if necessary
Event1:Timer Event
Action1:Save Result in DB, and Broadcast Result to all clients in the room
Spontaneous Action:Checking Next Item， Set Timer for the Item and Broadcast message
Checking Next Item
Spontaneous Action:Checking Next Item, if its null, send DELETE EVENT
End of Auction

Figure 15:


Onsite
Auction Scene System
Auction Scene System
Intranet/Internet
Backend
DB Node
DB Node
Web Server
Storage Node
Storage Node
Application Server
Storage Node
Storage Node
Load Balancer
Online Users
IOS App
Android App
Windows App
On site DVR System (from Model)
On site DVR System (from Model)
On site DVR System (from Model)
East Coast (DC)
West Coast (CA)
Web Server (from Model)
Application Server (from Model)
Load Balancer (from Model)
DB Node (from Model)
Storage Node (from Model)
Web Server (from Model)
Application Server (from Model)
Load Balancer (from Model)
DB Node (from Model)
Storage Node (from Model)
IOS App (from Model)
Android App (from Model)
Windows App (from Model)

Appendix 2:

User

Auction_user	
id	key
username	user name
password	encrypted password
role	administrator or general user
email	user email
phone	user phone number
real_id	use real id to register (safety)
reputation	reputation
creat_time	Timestamps
update_time	

Item

Auction_items	
id	item id
name	item name
category	help search
detail	description of the item
image	graph URL
room_id	belong to which room
price	current price
status	sold or not
price_user_id	to record which user give the highest price
creat_time	Timestamps
update_time	

Room

Auction_room	
id	room id
name	room name
des	description of the room
item_id	store items' id which will be sold in this room
user_num	how many users in the room
administrator_id	which administrator creates this room
status	whether the room is open
image	Image URL
creat_time	Timestamps
update_time	

Room User

Auction_room_user	
room_id	record who is in the room
user_id	
creat_time	Timestamps
update_time	

Order

Auction_order	
id	order id
user_id	who get this item
item_id	
status	whether paid or not
creat_time	Timestamps
update_time	

Table-relation: