Diff Kind of Booking Apps

- Movie → Seat
- Airplane → Seat
- Hotel → Room
- Train / Bus → Seat
- Restaurants → Seat

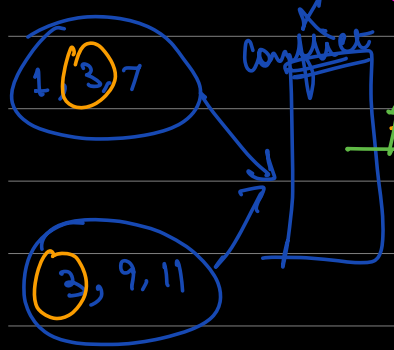
→ There is a problem of constraint / double booking.

Assign: Do the design for airplane mgmt system

Agenda

1. Motivation / Importance of Design
2. Design
   - a) Gathering Requirements
   - b) API Design
   - Architectural Design → Diff system
   - c) Class Diagram
   - d) Schema Design
   - e) Handling Concurrency

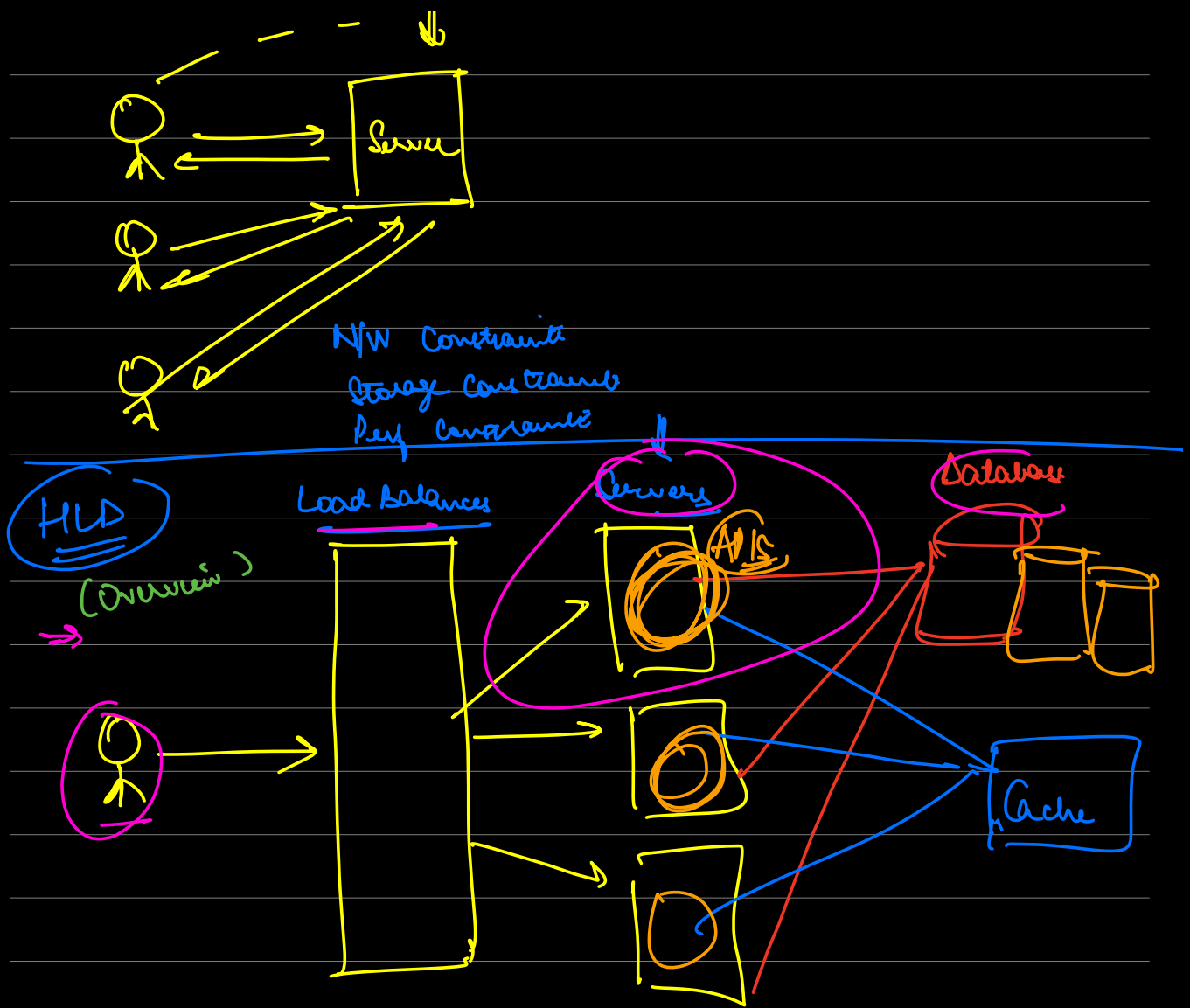
HLD / LLD



Context / Importance of Concepts

System Design

- HLD
  High level Design
  (Architectural Design)
- LLD
  Low level Design
  (Software Design)



LLD: How will my codebase be structured (Implementation)

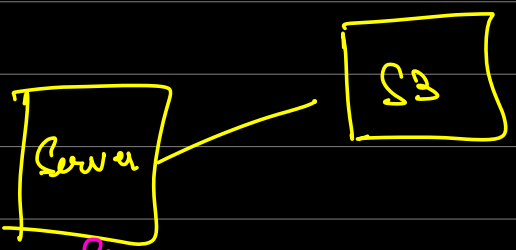
→ Object Oriented Design
- → What classes
- → attributes of classes
- → MVC
- → Concurrency
- → → DB design → tables
- → → API design

→ Write Code ✗

Amazon Senior Eng Manager

System Design

- LLD — Object Oriented Design
- HLD — Architectural Design



	FAANG	Startup
SWE 1	X	↳ (LLD)
SWE 2/L4	HLD (1 Round Optional)	↳ (LLD + HLD)
SWE 3 / L5	Must HLD	

Scaler LLD Curriculum

- OOPs (4 classes)
- SOLID Design Principles (2 classes)
- Design Patterns (5 classes)
- UML Diagrams
- MVC Pattern
- Movie Booking App

7 Design

→ Getting an overview

You know about the system (Have used the system as a user)

No idea
- Google
- Movie

(Can you please design a Movie ticket booking app? → Clarification → Handling Ambiguity + Get Knowledge

Theater API

Splitwise (Expense Sharing App)

System

Movie ticket booking:

You have an idea about functionalities of the system →

No idea

Overview =

→ you don't make any assumption

→ Get Buy in from interviewer before starting design

→ Also ask about type of design

ASSUME

Design for Backend System of a theatre that allows users to book seats

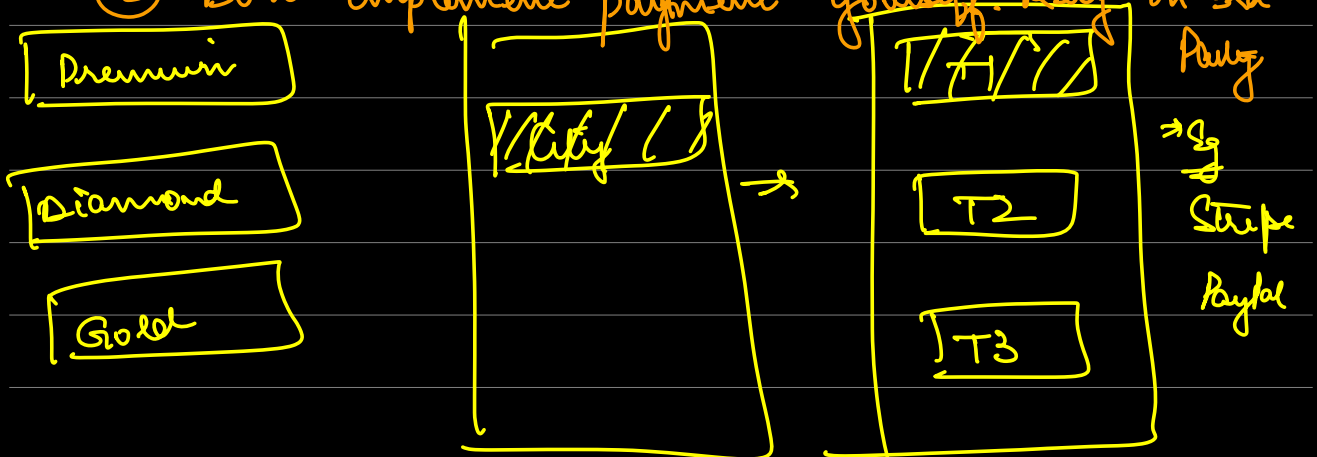
Gathering Requirements

Ask that q^n that can influence the design

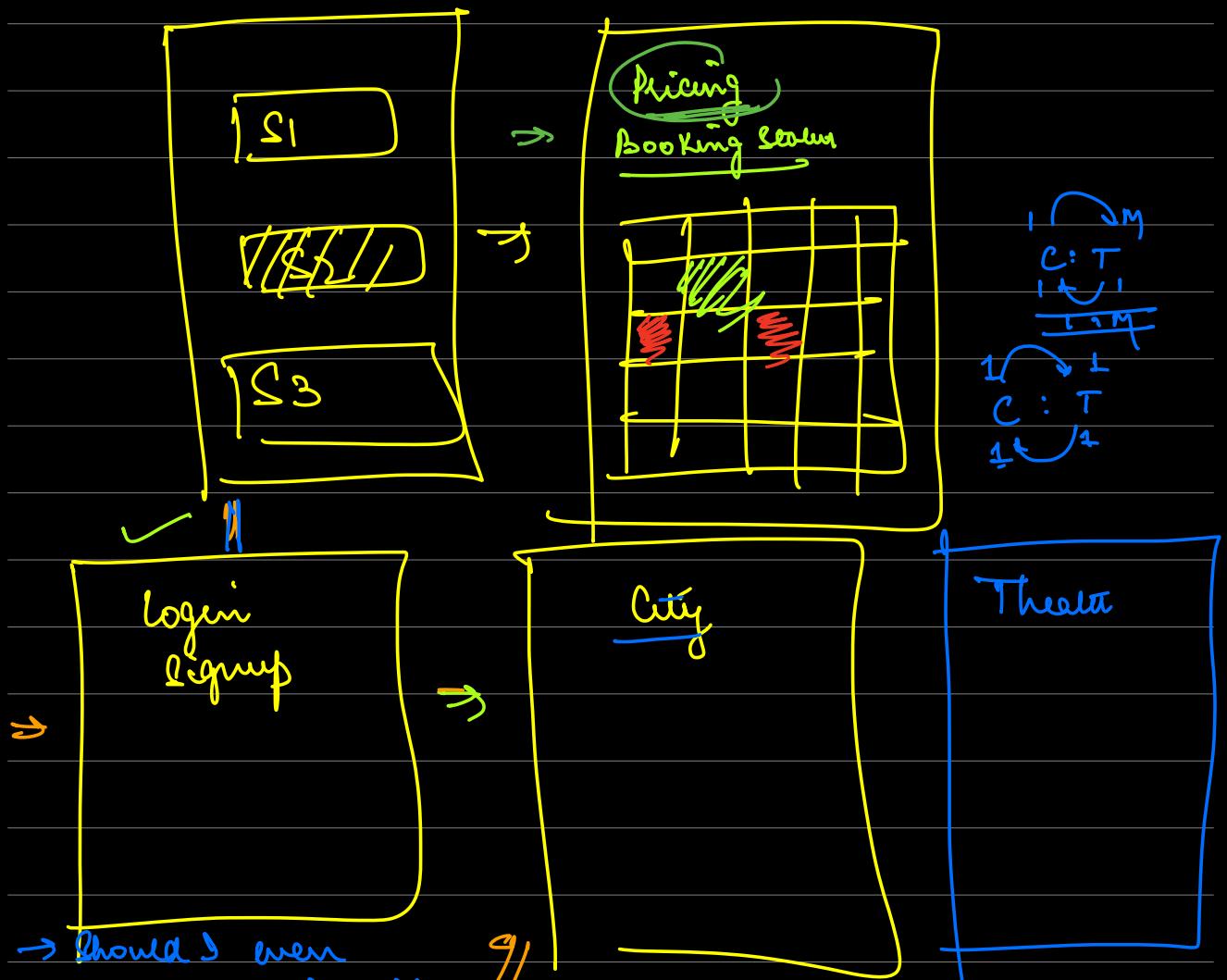
→ One Sided Comm^n ✗

→ Suggest ideas / features that you think must be there in a basic variant.

1. System should support multiple cities.
2. ~~Searching Movies~~ ✗
2. Should be able to see current status before booking
3. Diff pricing for diff type of seats for a show.
4. User will choose seats themselves.
5. User should have an ability to login / signup
6. Only support email / password for now.
7. Support cancellation
8. Don't implement payment yourself. Rely on 3rd Party



9. One theatre can have multiple screen



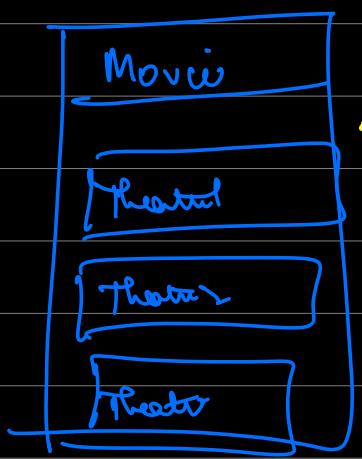
→ Should I even support this? ⇒ Yes

⇒ is it mandatory? ⇒ Yes.

⇒ what are user attributes
name, email, pass

⇒ Can user have diff roles
↳ admin / client ⇒ Yes

→ Can there be single theatre in multiple in a city
⇒ Multiple



← Do we want to support rating

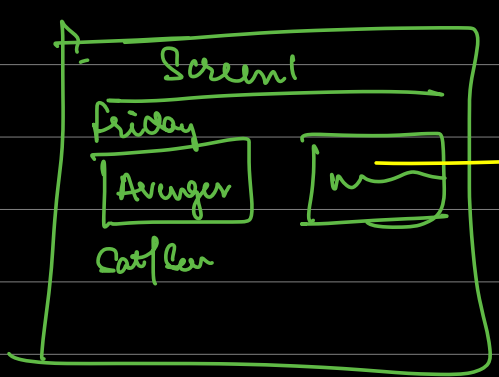
← Cast of movie

Gathering Req

1. Login and Sign up (name/email/pwd)
2. Support multiple cities
3. Support multiple theatres in a city
4. Support multiple screens in a theatre
5. Every screen will have diff types of seats.
6. Each seat may have diff price for a particular show

Movie

City
Theatre
Screen
→ Seat
Show
Ticket
Movie
Cast
name, role

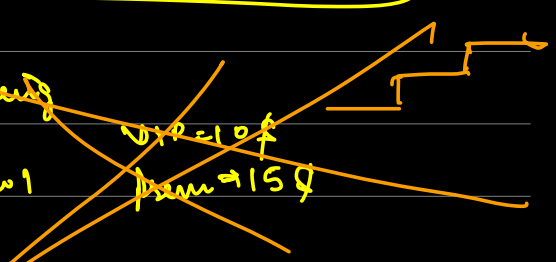


Static Pricing

S1 Show 1 VIP = 10$ Premium = 15$

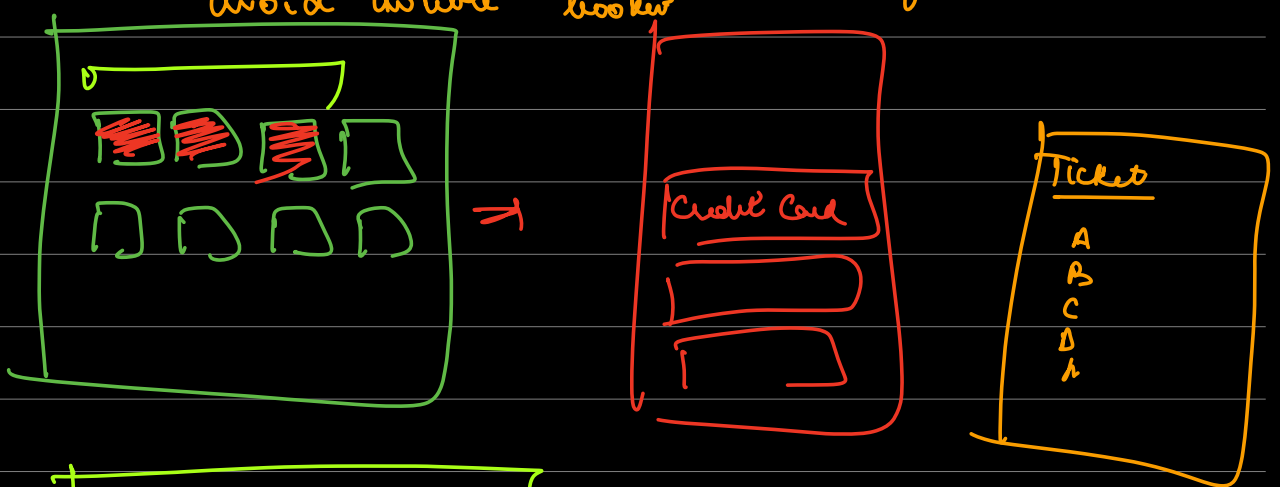
Dynamic Pricing

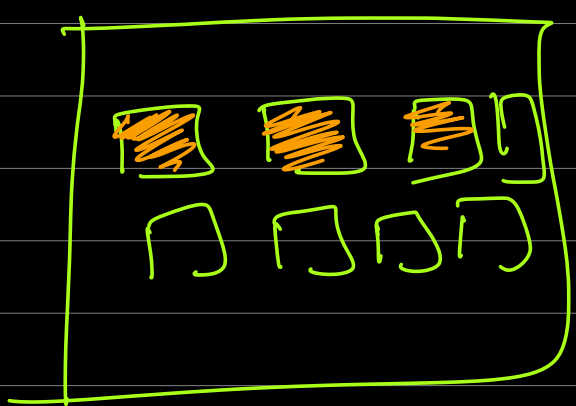
S1 Show 1



(7) We do have to support admin API

(8) Support for cancelling a ticket
(9) # of seats per booking ⇒ 10
(10) In one ticket only one show.
(11) Any seat that is available can be book
(12) A seat will remain on hold for 10 min to avoid double booker





APIs → User
→ Admin

On-Hold

API Design

Login

HTTPS

→ Hashed → Encrypt

/users/logIn

email
password

Hashed

/users/signUp

email
password
Name

GET /cities

GET /movies?city={}
GET /theatres?city={}
⇒ /movies?theatre={}

GET /movies/{id}

POST /bookings

{
showId: [
seats: [ ]
userId:
}

GET ⇒ Read
POST ⇒ Write

Search API → Get Post

Query limit 2048

Avengers

Show
- Movie
- Theatre
- Time
- Screen

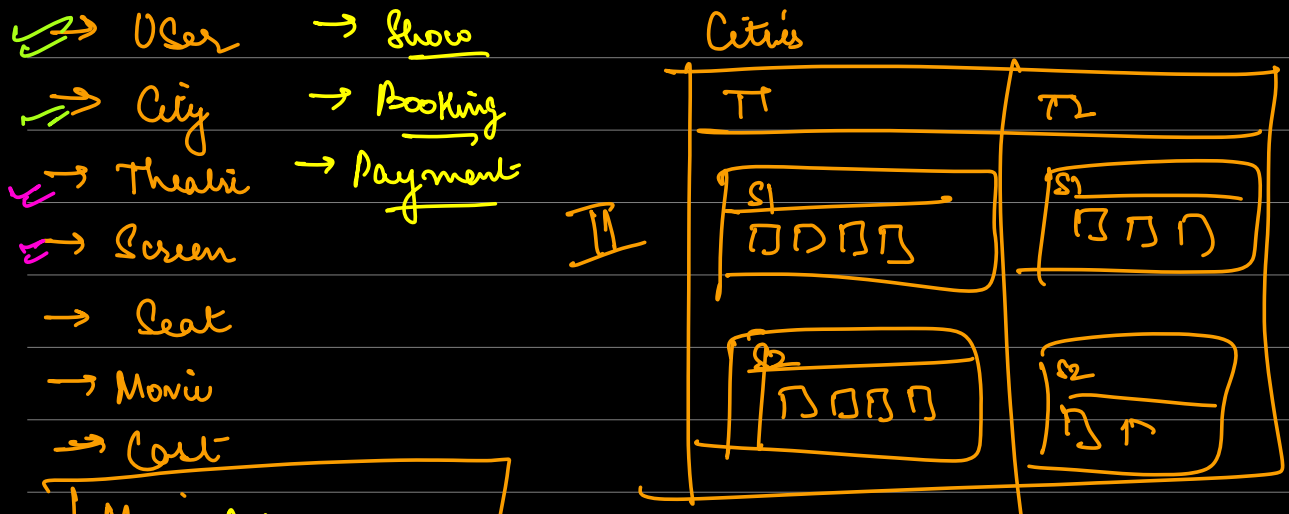
Class diagrams

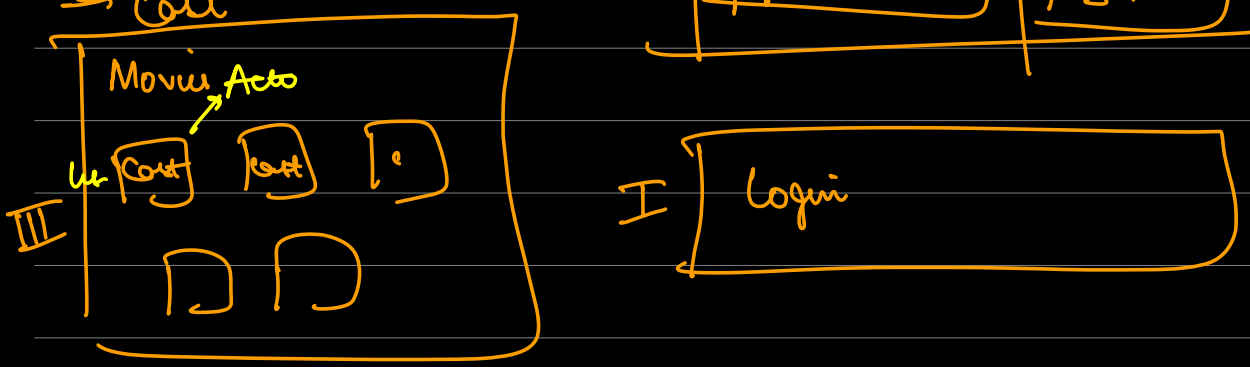
5 min 8:11 8:16

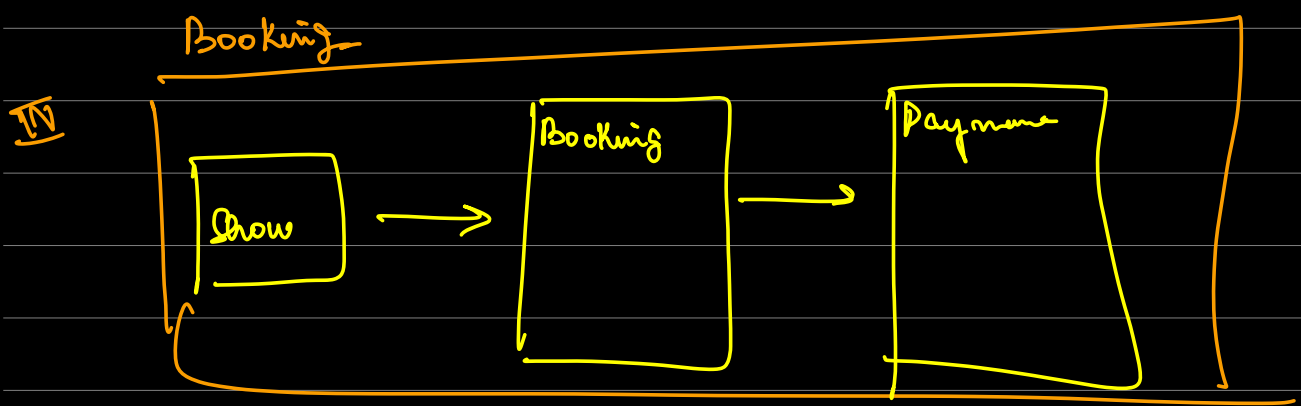
→ entities (dg things that have to store info about)

2 ways
- → find out all the nouns that are there in the req
- → visualize the system flow.

- User
- City
- Theatre
- Screen
- Seat
- Movie
- Cart
- Show
- Booking
- Payment







Show
Movie
Theatre
Screen
Time

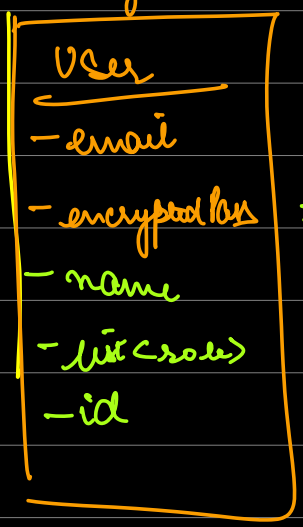
S1 = { A, T, S1, 7AM }

S2 = { A, T, S2, 8AM }

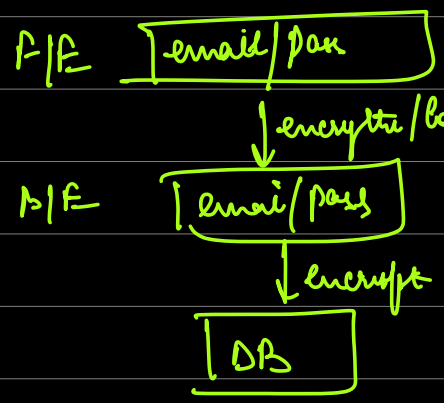
1 Movie taking 8hr. of screen

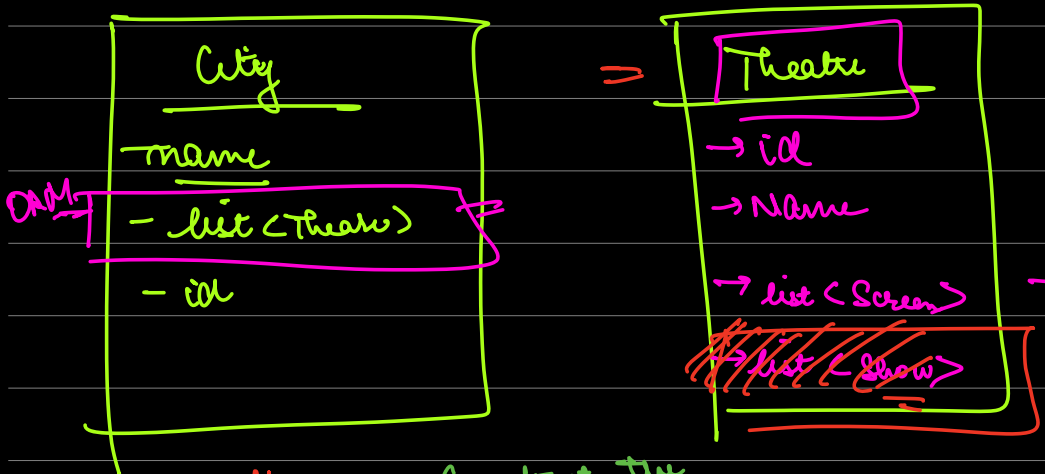
Shows	Avengers	IMAX	1PM	Screen1
Shows	Aven	IMAX	1PM	Screen2

Class Diagram



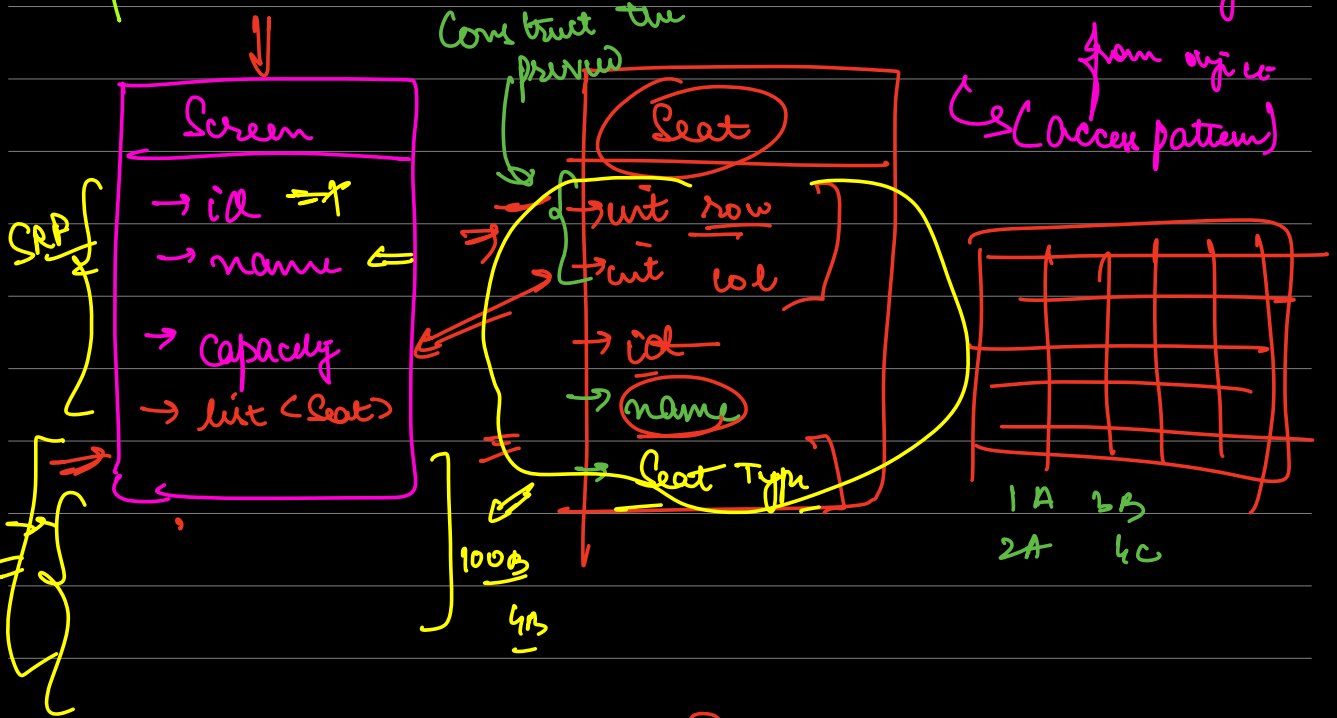
Role
- CUSTOMER
- ADMIN
- OWNER
- MODE





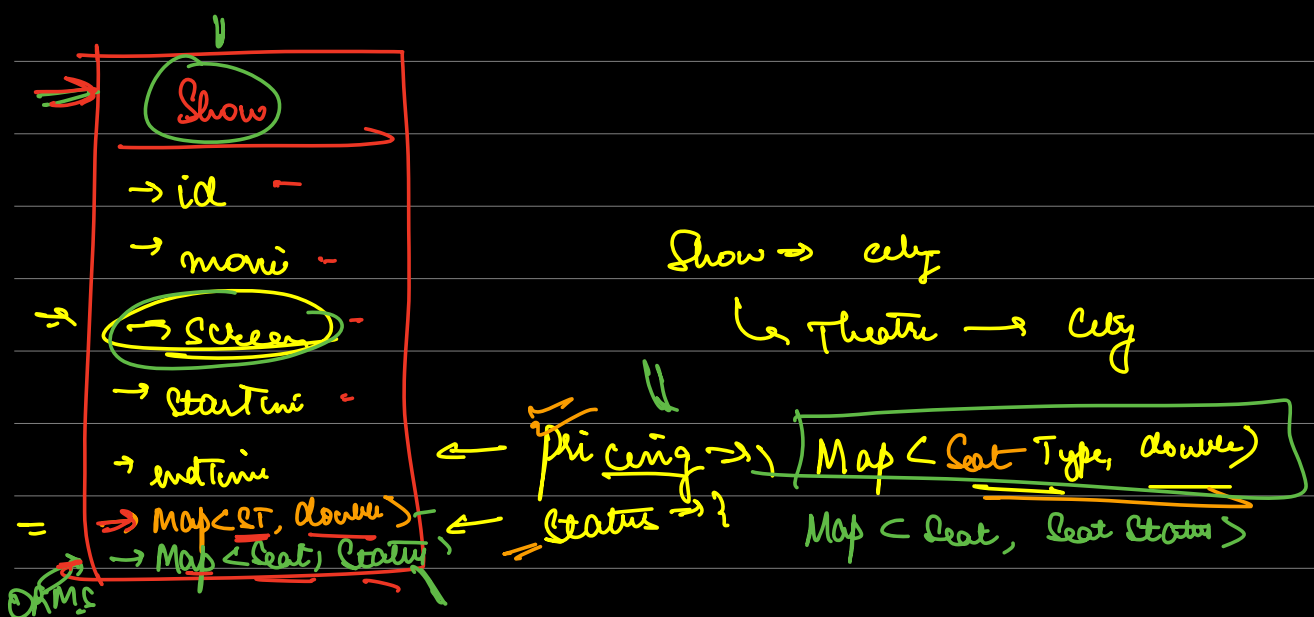
→ what info is needed to rep that object

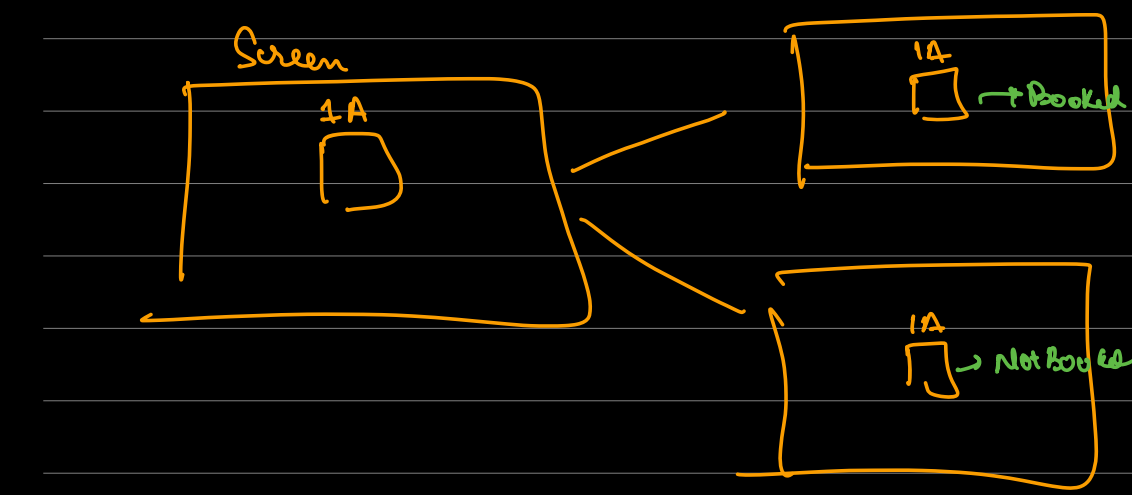
→ what info you might want to get from object (Access pattern)



GET /shows ? theatre_id = { }

→ get all shows for a theatre





Shows

id	movie_id	screen_id	start_time	end_time

SQL

Show_seat_type_price

id	show_id	seat_type_id	price

⇒

Show_seat_status

id	show_id	seat_id	status
1	1	200	
2	1	200	
3	1	200	

Redundancy ✗

Booking {

User.

show

list < Seat >

Amount

Status → Cancelled
Status → processing
Status → processed

time of Booking

}

How to Handle concurrent booking

2 users

→ 3 4 5 6 7 ⇐ U1

→ 6 7 8 4 ⇐ U2

Only one booking should succeed per seat

Use Database Locks

Assgn ‖ Transaction

(1) DB, Isolation levels

DBMS ⇒ ACID

- Only one person could go to the payment page for 1 seat at 1 time

Steps

⇒ As soon as someone selects seat and clicks continue. {1, 2, 3, 4} {3, 4, 5, 6}

a server request will go to BLOCK those seats for that server

{1, 2, 3, 4} {3, 4, 5, 6}

⇒ App server for each request creates a new DB transaction.

⇒ Select * from show_seat_status where seat_id IN (1, 2, 3, 4) for update;

↑ (3, 4, 5, 6

Transaction

{ select — for update; // for all the rows that will be fetched via the select, please lock them and don't allow anyone else to read them/ write them

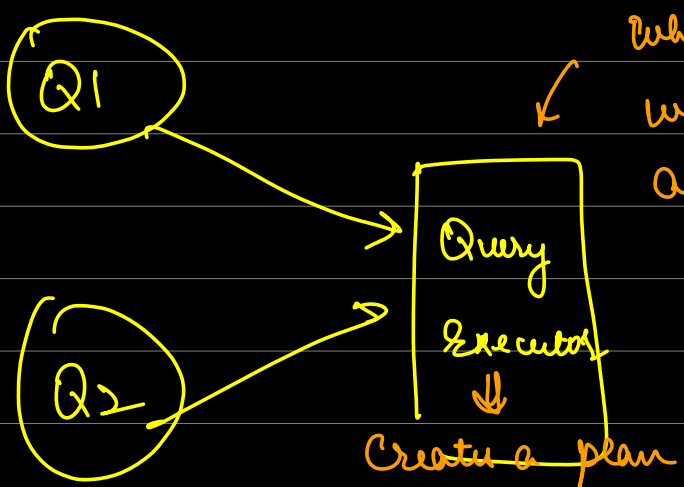
After a user selects the seats that they want to book

```
Select * from show_seat_status
where show_id = ( )
and seat_id IN ( )
for update;
```

→ check if the status of all seats is available

if status of all is not available
→ return failure

if status of all is available.
update the status of HOLD → timeout and I will start the payment



When see 2 transac^n
write Serializable
And updating same
things, it makes them
Single thread

Show - Seat - Status

Show_id	Seat_id	Status	time_when_set

OOP → ~~Head First~~ Java : The Complete Reference

DP → Refactoring. guru / Head first DP