Agenda : Code TicTacToe

1. Checking winner in O(1) ✓
2. MVC ✓
3. Start coding TicTacToe

Will start at 9:10PM

⇒ Machine coding problems have hidden DSA problem

⇒ MVC (Models, Views, Controllers)

⇒ Code Models of TTT and some tips

Implementing check winner in O(1)

→ after any move we will need to check if someone has won
  ↳ if yes, update state of game accordingly

→ how to check if someone won.

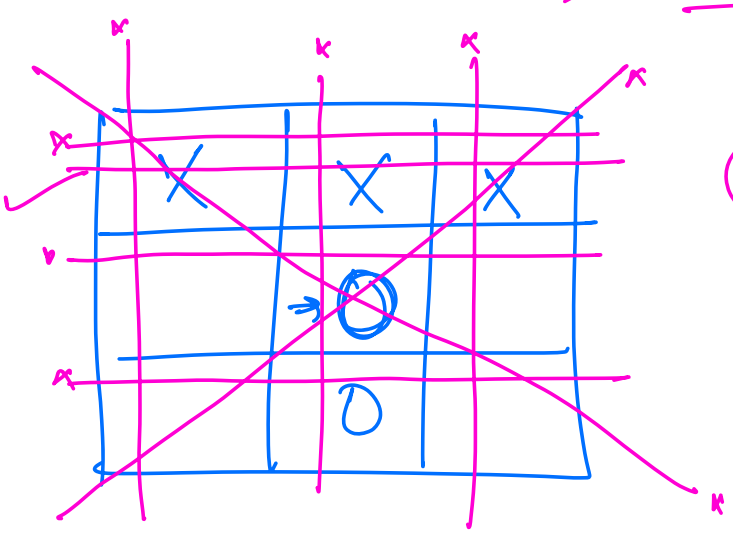
Assume

① There are only 3 ways to win:

→ Same symbol on complete row

→ ——— complete col

→ ——— complete diag.



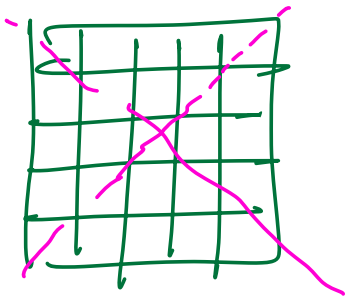
O (X) won

Approach 1 ⇒ TC: $O(N^3)$ SC: $O(1)$

```
for every symbol (O, X, --):          (n)
    for every row (i):                (n)
        for each cell of that row (j):(n)
            if cell[i][j].Symbol != Symbol
                break
```

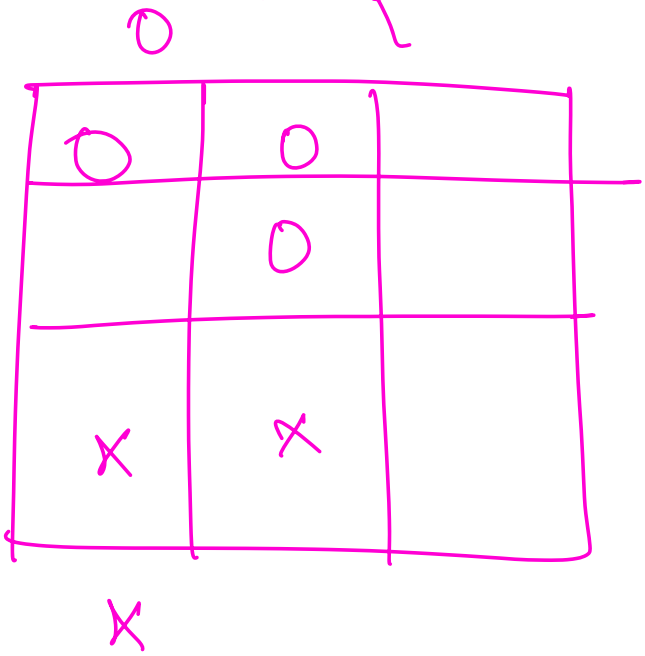
CheckWinner(Board);

```
for every col (j)
    for each cell of that col (i)
```



O(N) for each diag

O(N)



Observation: The person who made the last move is the only possible winner.

APPROACH 2

CheckWinner(Board, lastPlayer)

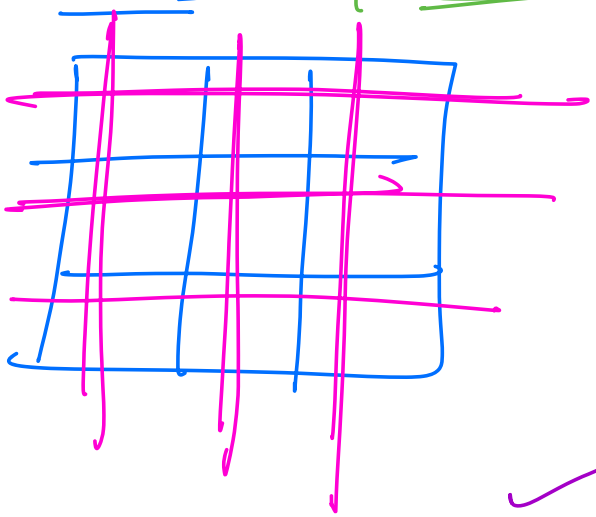
TC ⇒ O(N²)	SC ⇒ O(1)

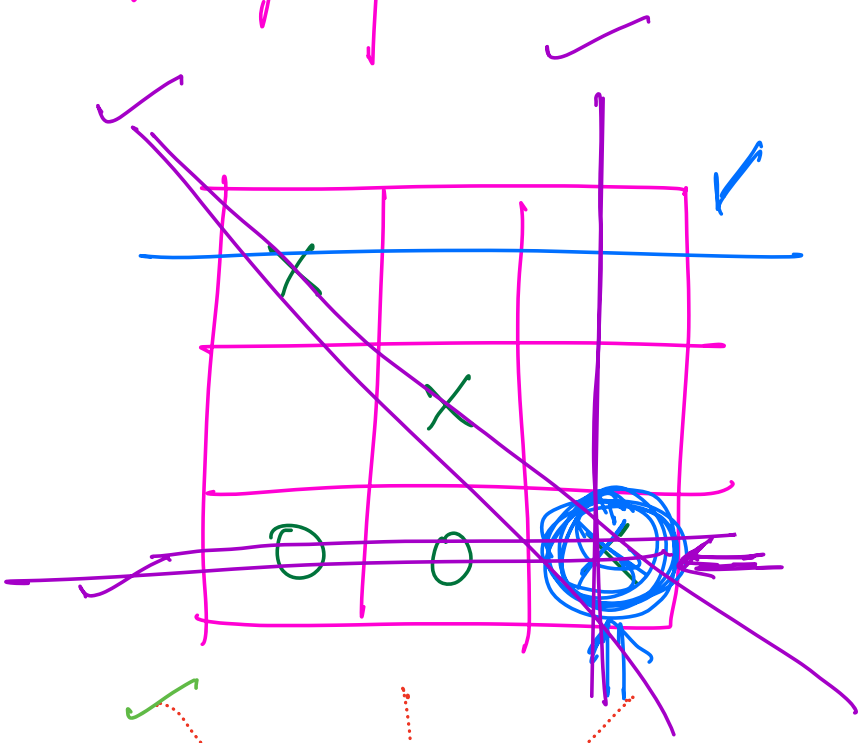
APPROACH 3

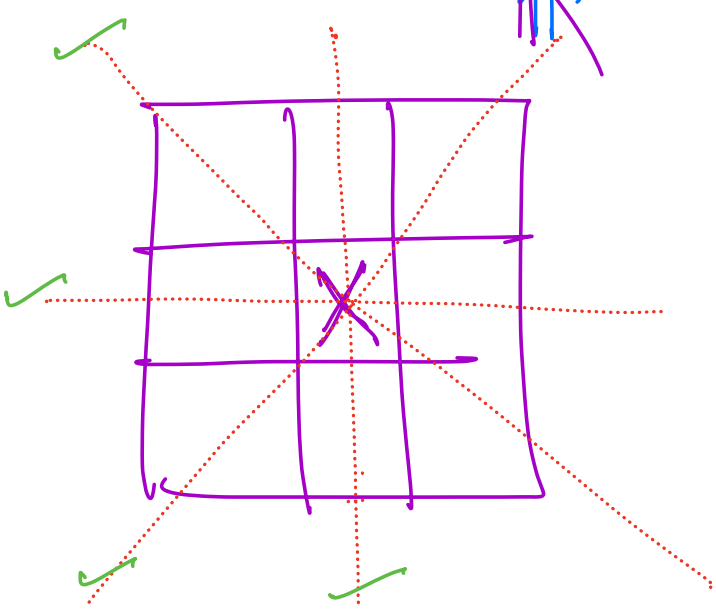
O(N) TC

O(1) SC

Why $O(N^2)$ ⇒ I was going through every cell of board ekbt.







move was made at (i,j)

(1) check if every cell i^{th} row contains same symbol ⇒ O(N)

(2) ——— j^{th} col ——— ⇒ O(N)

if (i,j) lies on left diag:

(3) ——— diag ———] O(N)

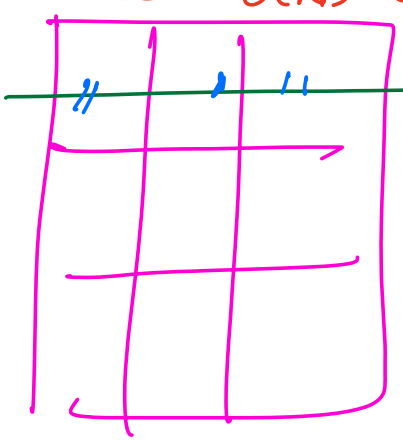
if (i,j) lies on right diag:

(4) ———] O(N)

⇒ TC ⇒ 4 * O(N) ⇒ O(N)

APPROACH 4 TC ⇒ O(1) =] O(1), O(N²)

SC ⇒ O(N) * O(N) ⇒ O(N²)



count

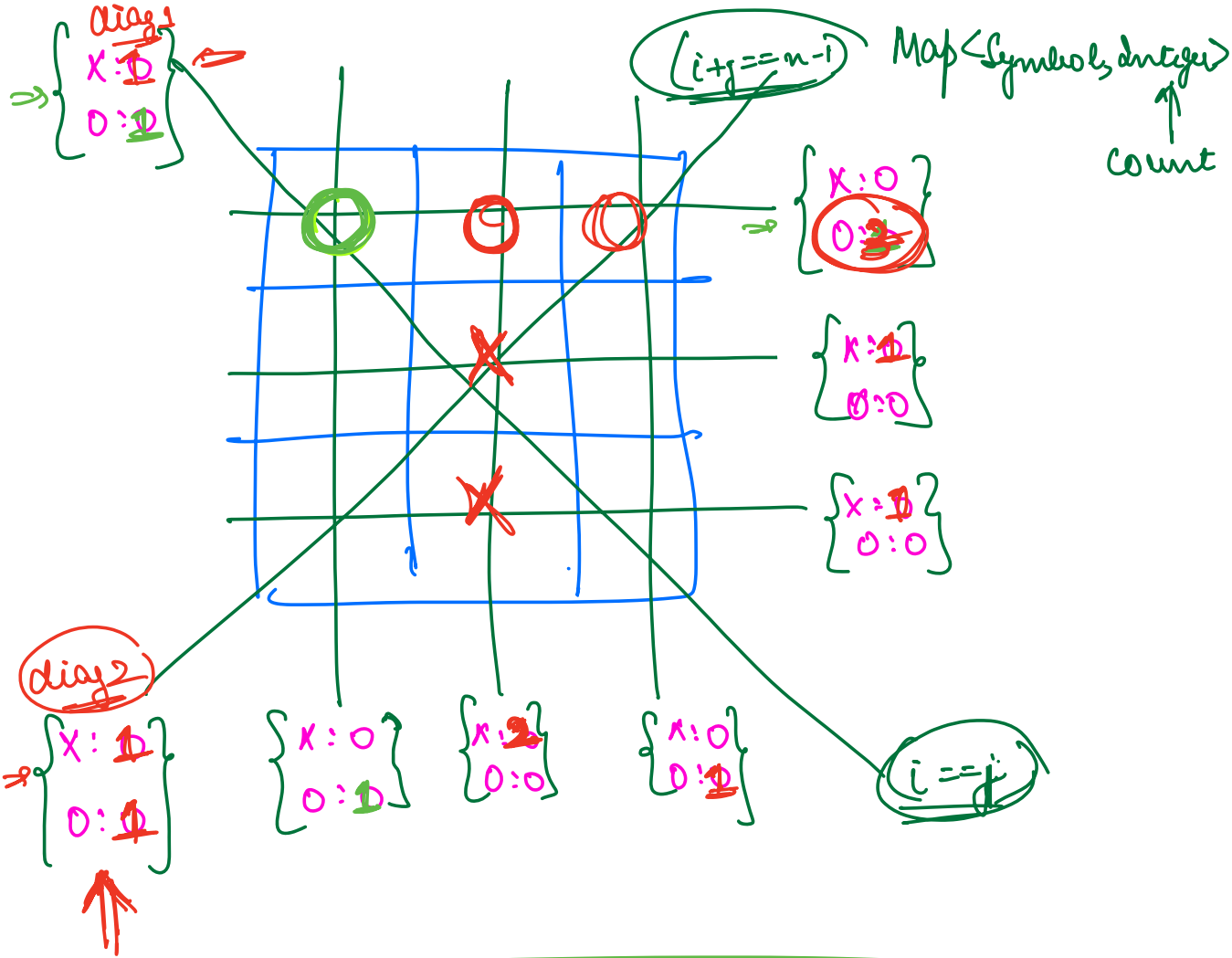
quantity(symbol)

==

quantity(row)

on(row)

== n



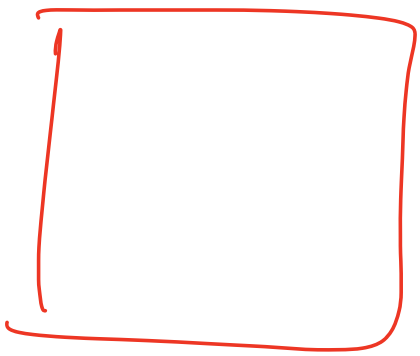
O(1) / move to update move

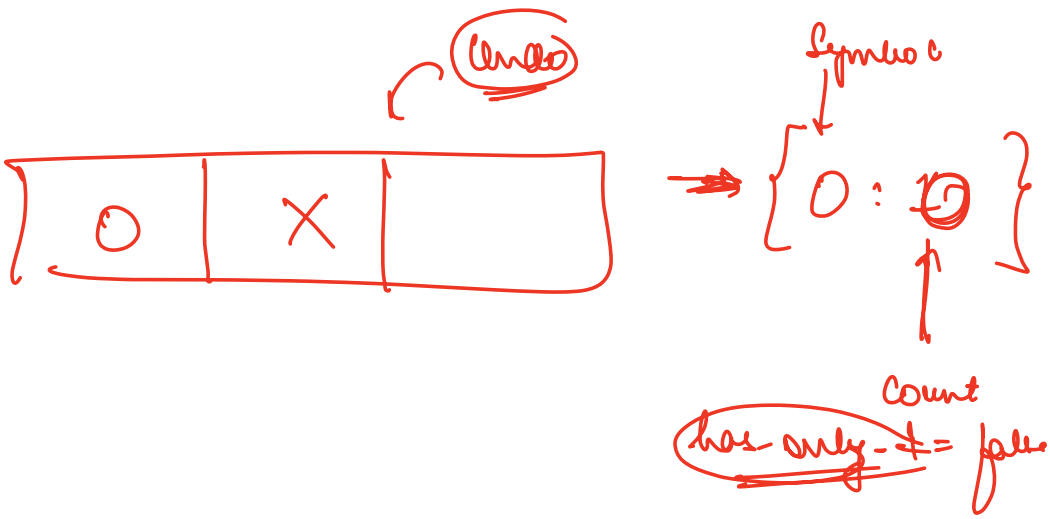
O(1) / check if value[key] == n

Space complexity ⇒ # maps * space/map

⇒ (n + n + 2) * (n+1)

≃ O(N²) ⇒ O(N)

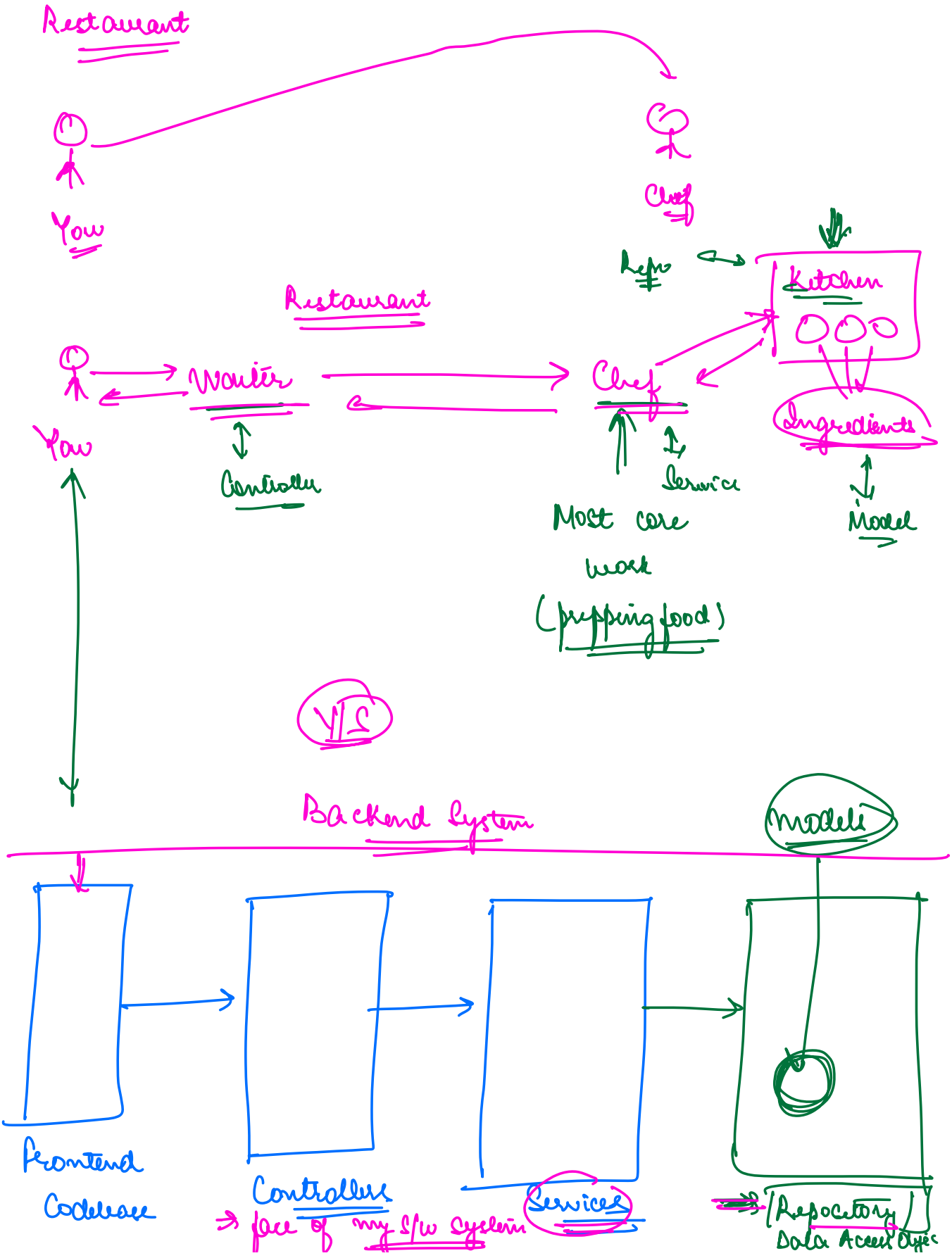




MVC Architecture

- M → Models
- V → Views
- C → Controllers

→ helps you on how to structure your codebase


Restaurant
You
Chef
Restaurant
You
Waiter
Chef
Kitchen
Ingredients
Controller
Service
Model
Most core work (prepping food)
V/S
Backend System
Models
Frontend Codebase
Controllers
⇒ face of my S/w system
Services
Repository / Data Access Objects

MVC ⇒ Structure your codebase into diff packages with each package having a very well defined responsibility

Controllers
- → Frontend code will call methods of controller classes
- → light-weight (no business logic)

Services
- → have business logic
- & do the real computation

Repository / DAO
- → will have methods to fetch data
- → how to talk to DB.
- → dont write code of SQL queries in services
- → write code in repositories.

class UserRepo {

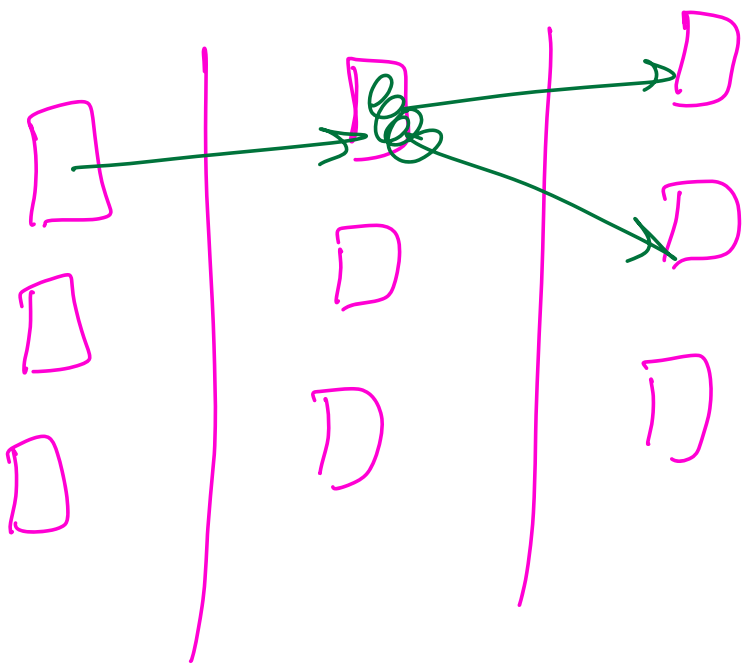
findUserById()

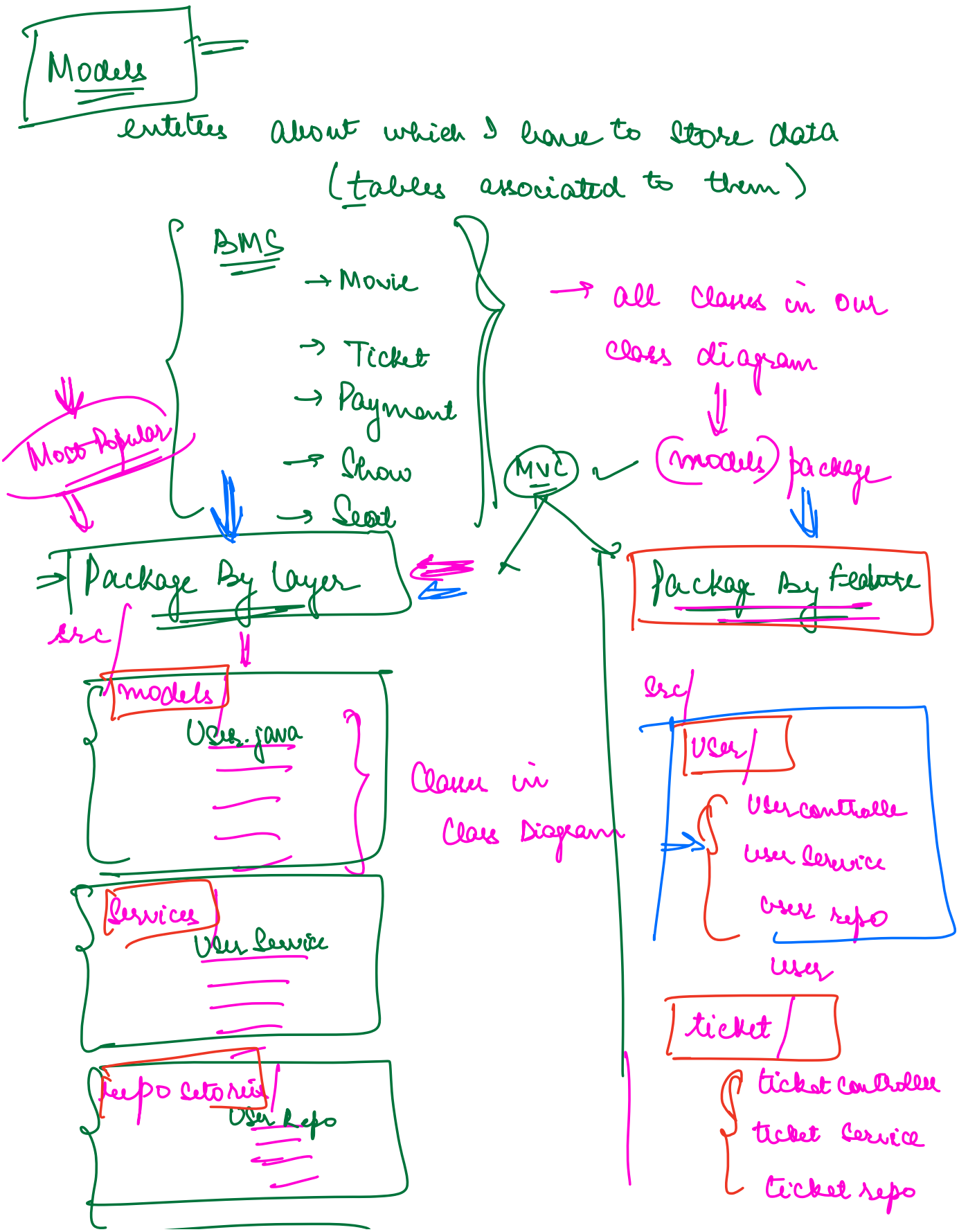
save(User)

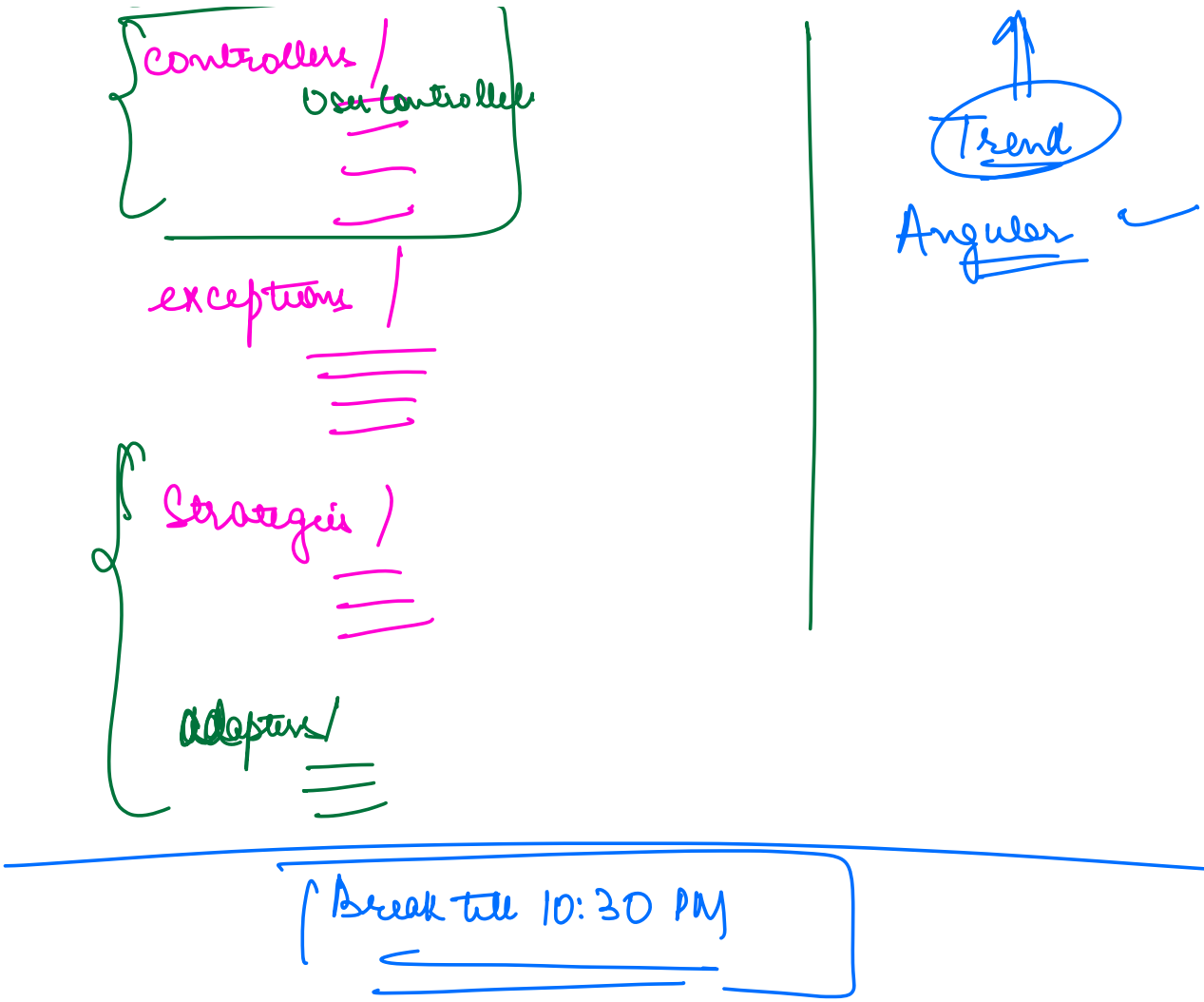
findUserByName(

}

refund: only till 3 hrs
unless you are a premium.
only till 1 hr before.




Models
entities about which I have to store data
(tables associated to them)
BMS
→ Movie
→ Ticket
→ Payment
→ Show
→ Seat
→ all classes in our class diagram
MovieBooker
MVC
(models) package
Package By Layer
Package By feature
src
models
Users.java
Classes in Class Diagram
src
User
UserController
UserService
UserRepo
User
ticket
TicketController
TicketService
TicketRepo
services
UserService
repositories
UserRepo


Controllers /
User controller
exceptions /
Strategies /
Adapters /
Trend
Angular
Break till 10:30 PM