Online Movie Ticket Booking Application Design:

ORACLE

By:

Prateek Suraksha Bhushan

Table Of Contents

Online Movie Ticket Booking Application Design: ORACLE

1. Problem Statement:

Objective:

Build a movie booking application like BookMyShow.com which can be used to:

- List movies with theatre, timing, language, rating, etc.
- Create user
- Book movie

Deliverables:

- Relational database schema in the form of DDL statements i.e. create a statement to persist the data
- Java or C/C++ language based application to interact with the data-store using APIs to perform CRUD (Create, Read, Update, Delete) operations

Design considerations:

- Scalability (e.g. multiple users, transactions etc.)
- Performance/Efficiency in terms of the amount of CPU, time and other resources consumed and response time of APIs
- Generic applicability of APIs
- Volume of data e.g. batch
- Backward compatibility i.e. clients should not be susceptible or strongly coupled to the APIs

----

Please submit your project report along with your deliverables, as described above, no later than **17 Nov 2022**.

2. Deliverables:

[A] Sample Data Definition Language for ticker booking application is provided in Section 3.

[B] C++ Ticket Booking Application Code:

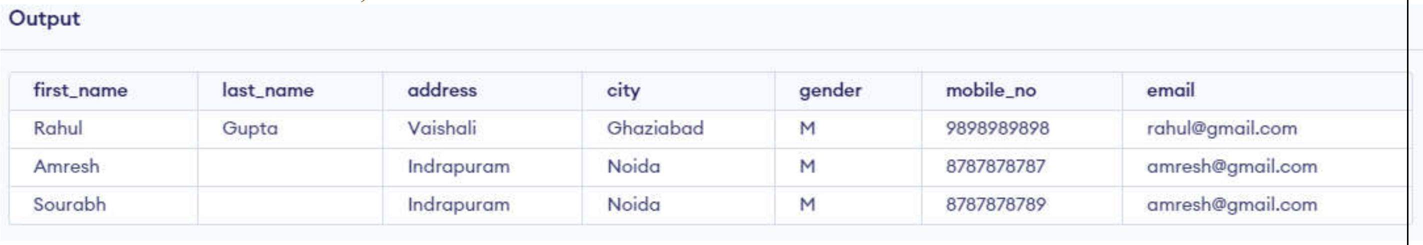
a. A Simple ticket booking application is created using C++ and STL.

b. Low Level Design which includes the logical flow of application, Activity Diagram and the UML Class Diagram are provided Section 4.

3. DDL for Movie Ticket Booking System:

a. User Table

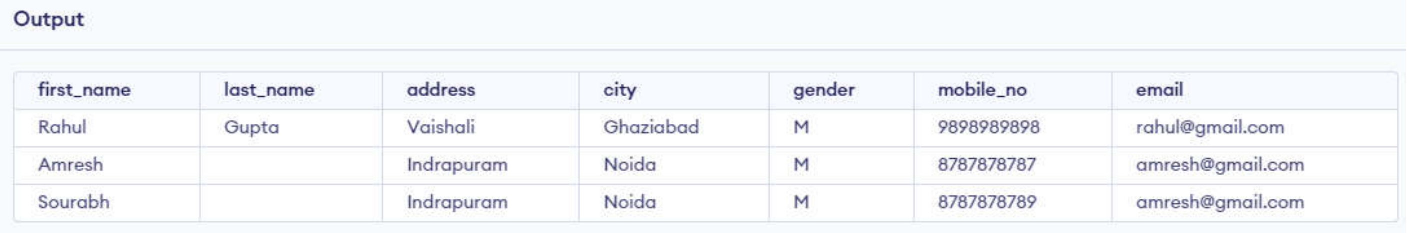
```
-- create a table
CREATE TABLE User  (
   first_name varchar(30) NOT NULL,
   last_name varchar(30),
   address varchar(80),
   city varchar(30) NOT NULL,
   gender varchar(1),
    mobile_no varchar(10) PRIMARY KEY,
    email varchar(30) NOT NULL

);
-- insert some values
INSERT INTO User VALUES ("Rahul","Gupta","Vaishali", "Ghaziabad", "M", "9898989898",
"rahul@gmail.com");
INSERT INTO User VALUES ("Amresh","", "Indrapuram", "Noida", "M",  "8787878787",
"amresh@gmail.com");
INSERT INTO User VALUES ("Sourabh","", "Indrapuram", "Noida", "M",  "8787878789",
"amresh@gmail.com");
-- fetch some values
>SELECT * FROM User;
```



Output

first_name	last_name	address	city	gender	mobile_no	email
Rahul	Gupta	Vaishali	Ghaziabad	M	9898989898	rahul@gmail.com
Amresh		Indrapuram	Noida	M	8787878787	amresh@gmail.com
Sourabh		Indrapuram	Noida	M	8787878789	amresh@gmail.com

```
>SELECT * FROM User where gender=="M";
```

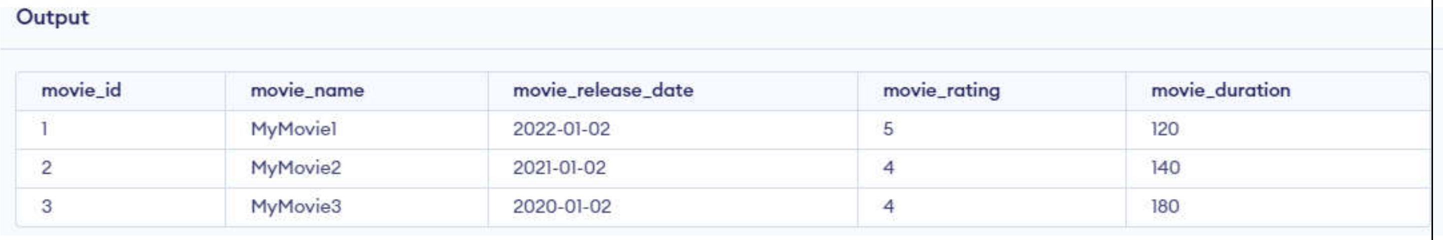


Output

first_name	last_name	address	city	gender	mobile_no	email
Rahul	Gupta	Vaishali	Ghaziabad	M	9898989898	rahul@gmail.com
Amresh		Indrapuram	Noida	M	8787878787	amresh@gmail.com
Sourabh		Indrapuram	Noida	M	8787878789	amresh@gmail.com

b. Movie Table

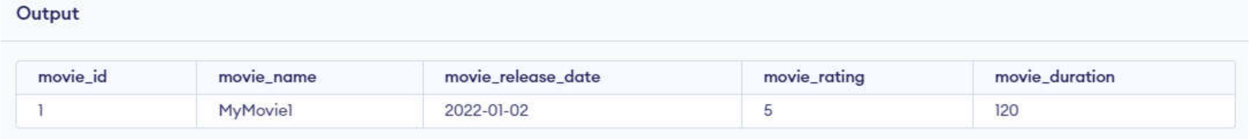
```
-- create movie table
        CREATE TABLE Movie (
          movie_id INTEGER  PRIMARY KEY,
          movie_name varchar(50) NOT NULL,
          movie_release_date DATE NOT NULL,
          movie_rating INTEGER,
          movie_duration INTEGER
          );
-- insert values into movie table
          INSERT INTO movie VALUES (1, "MyMovie1", "2022-01-02", 5, 120);
          INSERT INTO movie VALUES (2, "MyMovie2", "2021-01-02", 4, 140);
          INSERT INTO movie VALUES (3, "MyMovie3", "2020-01-02", 4, 180);
--run some queries
  SELECT * FROM Movie;
```



Output

movie_id	movie_name	movie_release_date	movie_rating	movie_duration
1	MyMovie1	2022-01-02	5	120
2	MyMovie2	2021-01-02	4	140
3	MyMovie3	2020-01-02	4	180

```
SELECT * FROM Movie WHERE movie_rating>4;
```

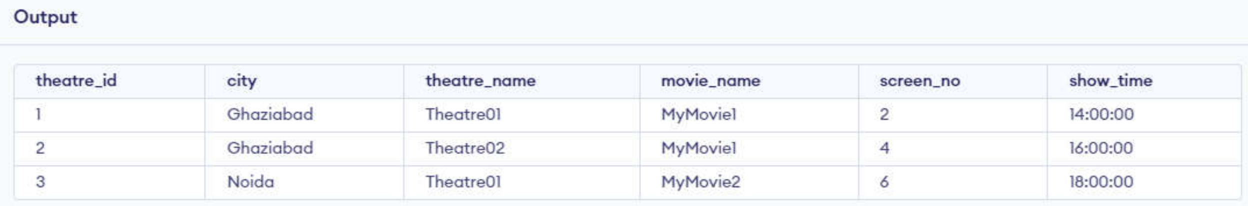


Output

movie_id	movie_name	movie_release_date	movie_rating	movie_duration
1	MyMovie1	2022-01-02	5	120

c. Theatre Table

```
--create Theatre table
        CREATE TABLE Theatre (
         theatre_id INTEGER,
         city varchar(30) NOT NULL,
         theatre_name varchar(50) NOT NULL,
         movie_name varchar(50) NOT NULL,
         screen_no INTEGER NOT NULL,
         show_time TIME NOT NULL,
         PRIMARY KEY (movie_name, show_time)
         );
--Insert few values
INSERT INTO Theatre VALUES (1, "Ghaziabad","Theatre01","MyMovie1", 2, "14:00:00");
INSERT INTO Theatre VALUES (2, "Ghaziabad","Theatre02","MyMovie1", 4, "16:00:00");
INSERT INTO Theatre VALUES (3, "Noida","Theatre01","MyMovie2", 6, "18:00:00");
--Display values
  SELECT * from Theatre;
```



Output

theatre_id	city	theatre_name	movie_name	screen_no	show_time
1	Ghaziabad	Theatre01	MyMovie1	2	14:00:00
2	Ghaziabad	Theatre02	MyMovie1	4	16:00:00
3	Noida	Theatre01	MyMovie2	6	18:00:00

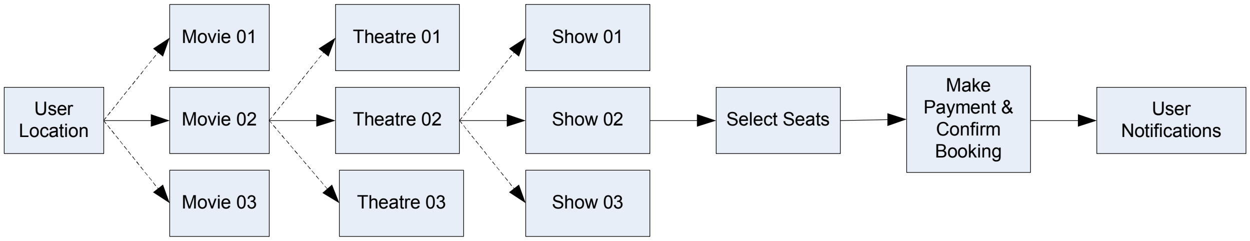
d. Seats Table

```
CREATE TABLE Seats(
  id INTEGER PRIMARY KEY,
  category ENUM('SILVER', 'GOLD', 'PLATINUM'),
  sear_available bool,
  price int
  );
INSERT INTO Seats VALUES (1, 1,0, 200);
INSERT INTO Seats VALUES (2, "GOLD",1, 200);
INSERT INTO Seats VALUES (3, "PLATINUM",0, 200);
```

e. Booking Table

```
CREATE TABLE Booking AS (
 SELECT Theatre.theatre_id, Theatre.screen_no, Seats.id, User.mobile_no
        from User
        INNER JOIN Theatre
        ON User.city==Theatre.city
        INNER JOIN Movie
        ON Theatre.movie_name=Movie.movie_name
 );
```

4. Low Level Design of BookMyShow Application:



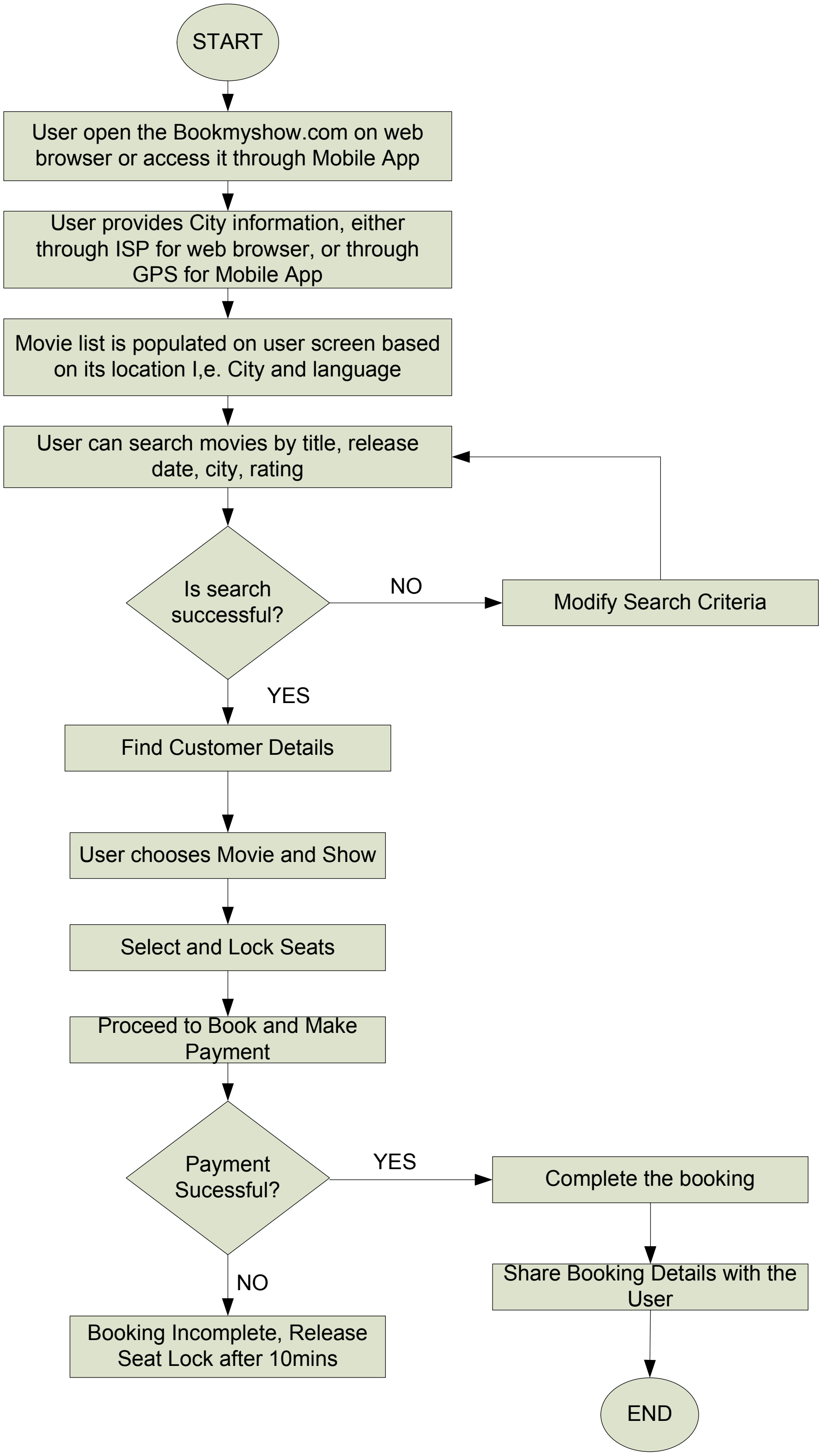
Conceptual Sequence of Bookmyshow application for a user

A movie booking application is tightly coupled with the City in which user is using the app.

General requirements for the user may be summarized as:

a. Select “City”, (may get information directly from mobile GPS for app and ISP in case of browser)
b. All the movies shall be listed w.r.t that City on user’s home screens along with the language and reviews information.
c. User shall select the Movie, he/she interested in.
d. This should display all the theatres displaying that movie along with show timings.
e. User shall select the show timing and proceed towards seat selection.
f. User shall be able to select the seat and proceed to make payment to complete the booking.
g. User can make payments through credit card, debit card, UPI, netbanking.
h. Once payment is done, and booking is confirmed, it shall be notified to user.

Activity Diagram of Online Ticket Booking System will look like:


START
User open the Bookmyshow.com on web browser or access it through Mobile App
User provides City information, either through ISP for web browser, or through GPS for Mobile App
Movie list is populated on user screen based on its location I,e. City and language
User can search movies by title, release date, city, rating
Is search successful?
NO
Modify Search Criteria
YES
Find Customer Details
User chooses Movie and Show
Select and Lock Seats
Proceed to Book and Make Payment
Payment Sucessful?
YES
Complete the booking
Share Booking Details with the User
END
NO
Booking Incomplete, Release Seat Lock after 10mins

Activity Diagram

Core Entities(Objects and Classes)

1. Movie
2. City
3. Theatre
4. Screen
5. Show
6. Seat
7. Booking
8. Payment

UML Diagram[File Attached at end]

5. High Level Design:

Design Considerations:

1. User can access OnlineTicketSystem (bookmyshow.com) through website or bookmyshow app.
2. In both cases, the URL is converted to IP address by DNS and requested is routed to bookmyshow servers.
3. To make system; there should be redundancy in servers so that single point of failure can be avoided.
4. As the number of request increases, the system is scaled. The task of web server is separated from Data Server. Load Balancer are introduced so that complete load is not shifted to only one server.
5. Since the system is dependent of location, to serve requests faster, Content Deliver Network and Cache can be introduced.
6. Further, scaling can be done by establishing data centers which consists of redundant web server and data server and are configured to serve requests based on geographical location of the requests.
7. Further, module to maintain and report errors, failure, request logs, CPU, Network and other modules/interfaces/modules shall be deployed to keep the performance in check.

Below is a System Architecture, for scaling the system from nascent stage, serving few hundreds of requests to millions of requests.

Step A: Basic Design

A basic, minimalistic high level design would consist of a single server, where user request from web browser or mobile app is routed to DNS, and then to web server to fetch HTTP/JSON response as shown in Figure 1. Single server is taking care of all the read and write requests, and maintaining the theatre and movie database.

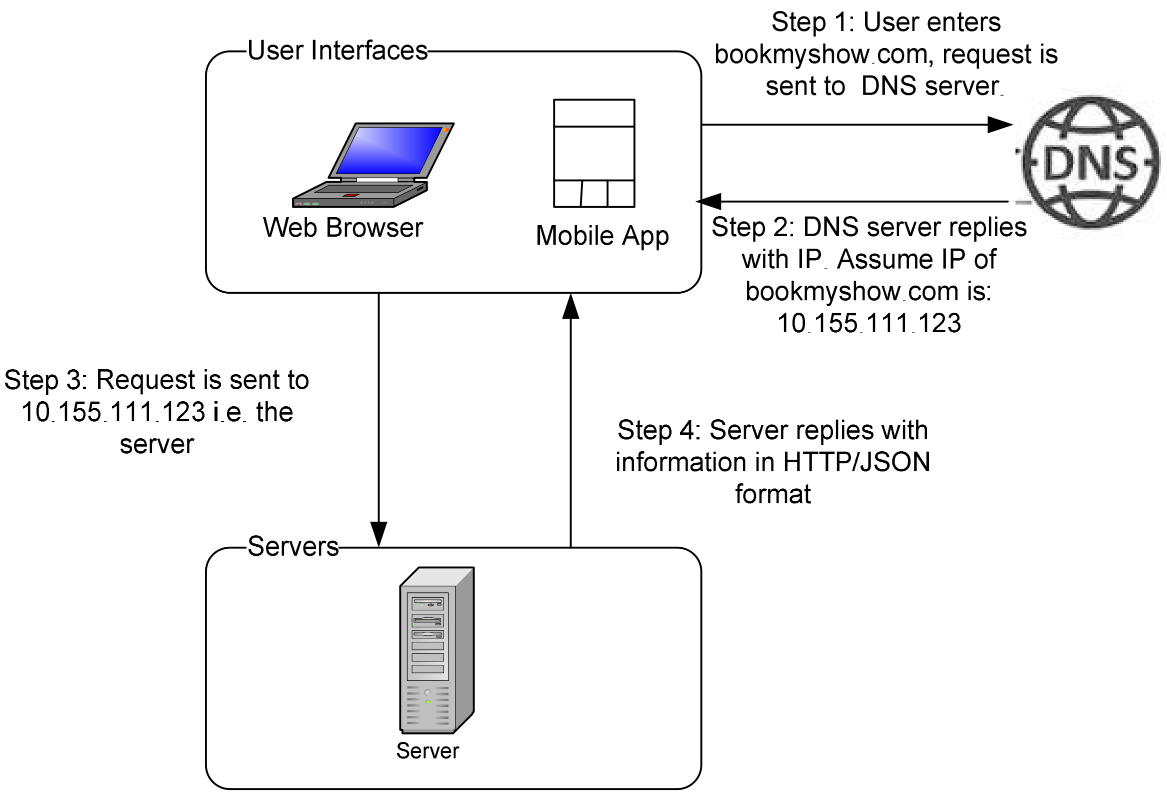

User Interfaces
Web Browser
Mobile App
Step 1: User enters bookmyshow.com, request is sent to DNS server.
DNS
Step 2: DNS server replies with IP. Assume IP of bookmyshow.com is: 10.155.111.123
Step 3: Request is sent to 10.155.111.123 i.e. the server
Step 4: Server replies with information in HTTP/JSON format
Servers
Server

Figure 1: Basic Simple Configuration with Single Server to handle small number of requests

Figure 1 Basic Design

Step B:

As the number of requests increases, the load on server would also increase. The request to server prominently consists of web/mobile traffic called as web tier and the other is database related traffic which is known as data tier. Separating the two types of traffic would allow independent scaling of servers.

Thus, web server and database servers are separated as showing in Figure 2.

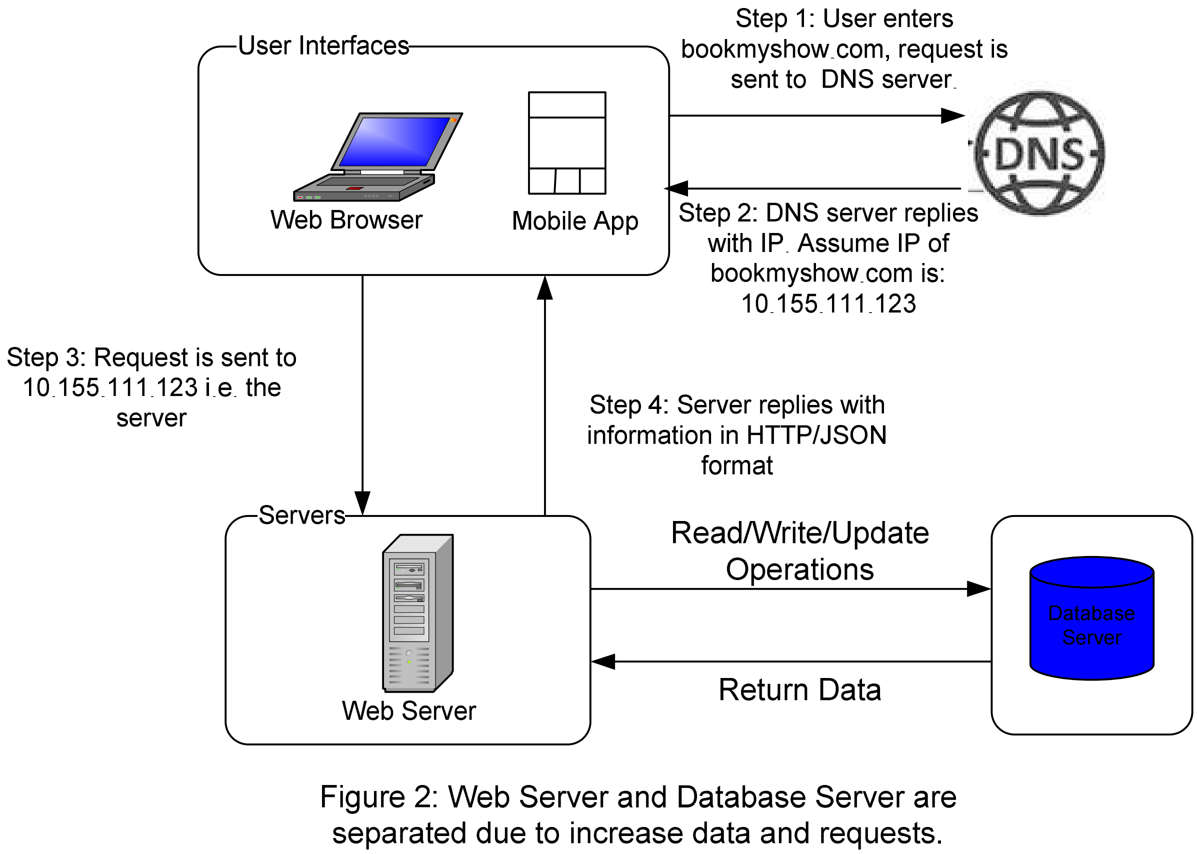

User Interfaces
Web Browser
Mobile App
Step 1: User enters bookmyshow.com, request is sent to DNS server.
DNS
Step 2: DNS server replies with IP. Assume IP of bookmyshow.com is: 10.155.111.123
Step 3: Request is sent to 10.155.111.123 i.e. the server
Step 4: Server replies with information in HTTP/JSON format
Servers
Web Server
Read/Write/Update Operations
Return Data
Database Server
Figure 2: Web Server and Database Server are separated due to increase data and requests.

Figure 2 Functionality of Web Server and Database Server Seperated

Here, database used can be a RDBMS as data is structured. Relational database like Oracle, MySQL, MariaDB etc can be used in this case.

For example: Data/Tables to be stored shall primarily consist of

a. User Data
b. Theatre Data
c. Movies Data

<u>Step C:</u>

Introducing Load Balancers, additional web servers and replicated databases:

The architecture in Step B illustrates that users are connected directly to the server, but, what will happen if server goes offline? Or suddenly there are too many requests which server may not able to process simultaneously. Moreover, there is only single database, which makes complete system vulnerable to single point failure. If database fails, the user credentials would not be verified, theatre shows and seat status would not available, i.e. it’s a complete system failure.

To establish a reliable, efficient and scalable system, following elements shall be added in Step B architecture.

a. Load Balancer is introduced, so as to evenly distribute traffic among active servers. The distribution of traffic can be defined through relevant policy also. (Like 60-40 traffic etc).
b. Another web server is introduced to handle large number of user requests.
c. Databases are replicated, where one is master database and rest are slave databases. The master database would perform all write and update operations and all read operations would be performed on slave databases. All the databases would be synchronized, i.e. wherever master database is updated, it will send the updated information to slave databases too. i.e. Whenever new user is created, or theatre information is updated, booking is completed etc, the master database will be updated which in turn updates the slave databases.

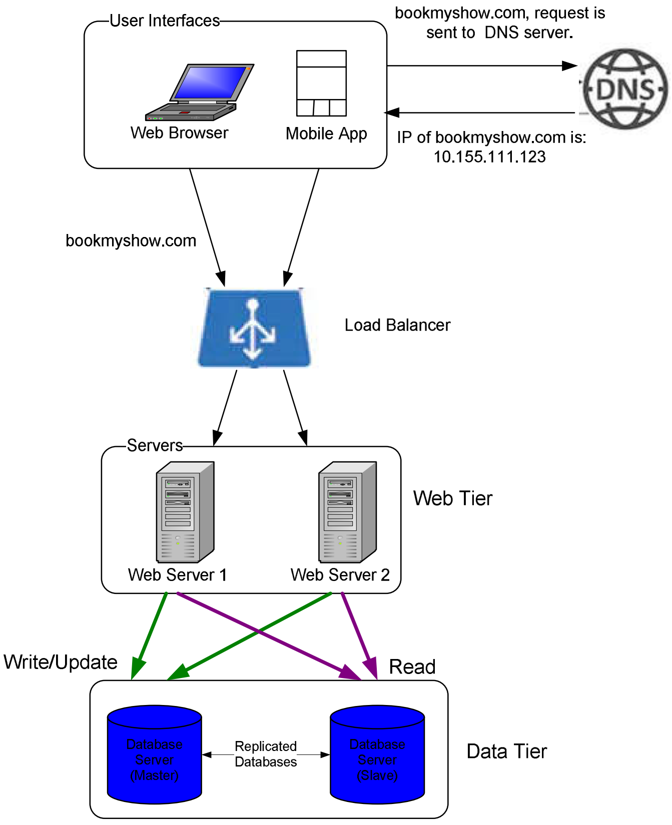


Figure 3: System Design after introducing Load Balancer, Web Server and Database Replication

Figure 3 System Design after introducing Load Balancer, Web Server and Database Replication

Step D:

Introducing Cache and Content Delivery Network (CDN)

Cache: Caches is a temporary storage which stores the frequently used data. If a information is needed very frequently, and every time the information is accessed from database/server, it will be time consuming and make the system slow. In order to increase speed of accessing frequently used information, the information can be stored in cache.

For example, recently launched movies information and ratings can be kept in cache, or information regarding movies which are booked most recently can be kept in cache.

Content Delivery Network:

A CDN is a network of geographically dispersed servers which is used to deliver static content. CDN servers cache static content like images, videos, CSS, JavaScript files, etc.

CDN working: when a user visits a website, a CDN server closest to the user will deliver static content. Intuitively, the further users are from CDN servers, the slower the website loads.

Since, there is a tight coupling between the theatre listing and cities, the nearest CDN can deliver content based on the preference of users belonging to that city. Thus will reduce page load time and enhance user experience.

After including cache and CDN the architecture would look like Figure 4.

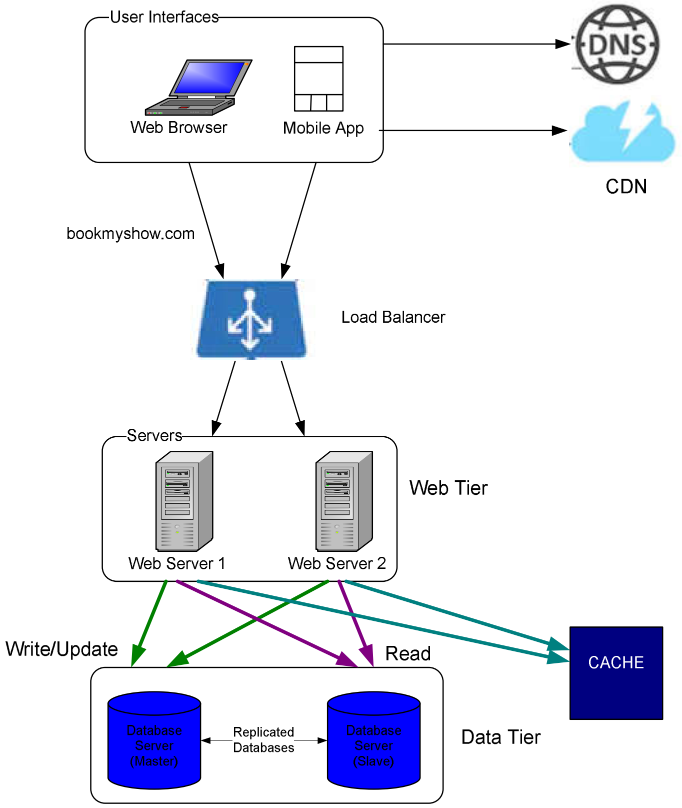


Figure 4: System Design after introducing Cache and Content Delivery Network (CDN)

Figure 4 Introducing Cache and CDN

Step E:

Web Tier, Data Tier and Cache all together form a Data Center. There can be number of geographically dispersed data centers. Load balancer can be configure to segregate user request based on geography and route them to relevant data center.

With increase in user base and size of system, tools are required to log and monitor system performance metrics and automate error reporting, built, test and deploy tools.

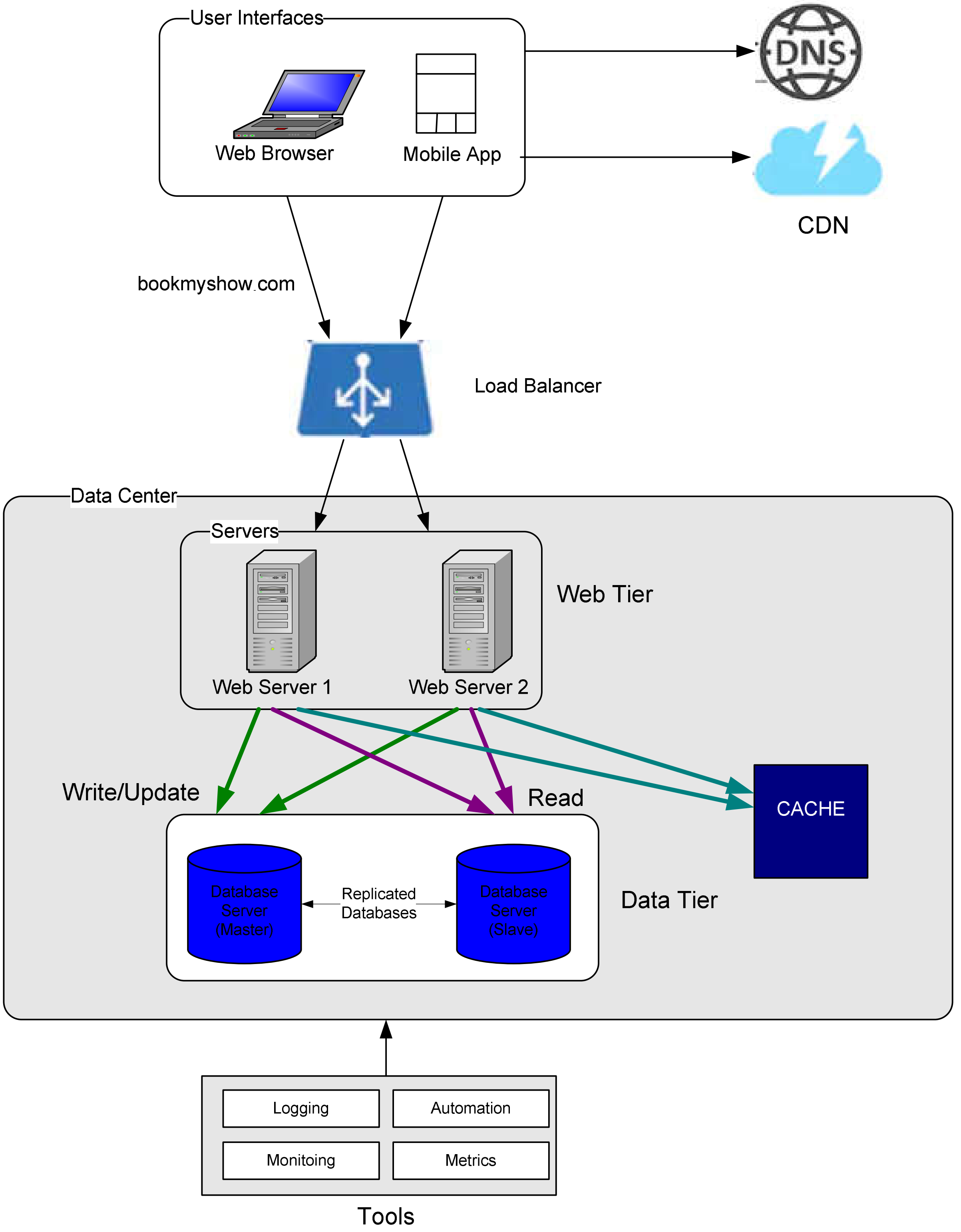


Figure 5: System Design with Data Center and Tools

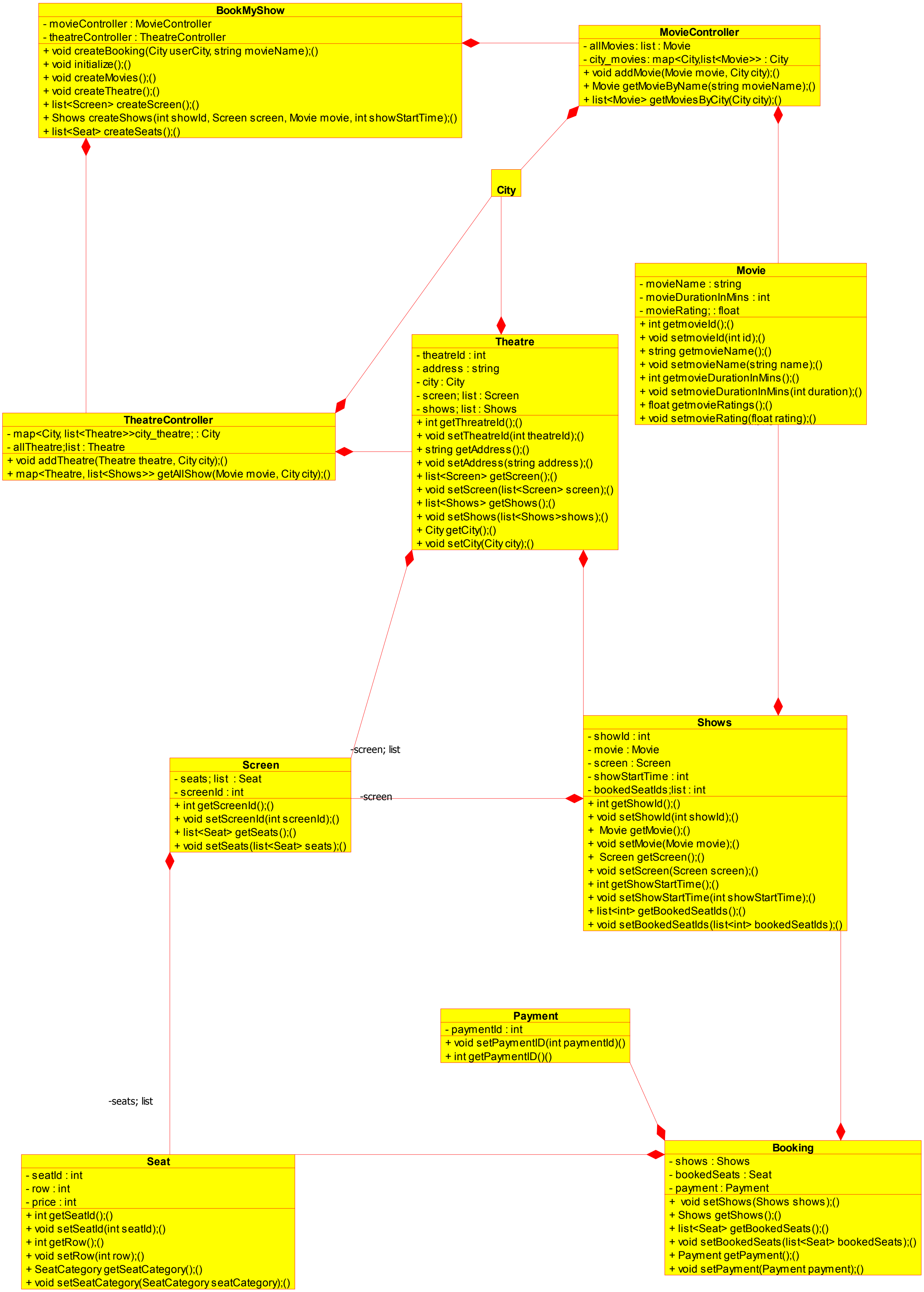

BookMyShow
- movieController : MovieController
- theatreController : TheatreController
+ void createBooking(City userCity, string movieName);()
+ void initialize();()
+ void createMovies();()
+ void createTheatre();()
+ list<Screen> createScreen();()
+ Shows createShows(int showId, Screen screen, Movie movie, int showStartTime);()
+ list<Seat> createSeats();()
MovieController
- allMovies : list : Movie
- city_movies: map<City,list<Movie>> : City
+ void addMovie(Movie movie, City city);()
+ Movie getMovieByName(string movieName);()
+ list<Movie> getMoviesByCity(City city);()
City
Movie
- movieName : string
- movieDurationInMins : int
- movieRating; : float
+ int getmovieId();()
+ void setmovieId(int id);()
+ string getmovieName();()
+ void setmovieName(string name);()
+ int getmovieDurationInMins();()
+ void setmovieDurationInMins(int duration);()
+ float getmovieRatings();()
+ void setmovieRating(float rating);()
Theatre
- theatreId : int
- address : string
- city : City
- screen; list : Screen
- shows; list : Shows
+ int getThreatreId();()
+ void setTheatreId(int theatreId);()
+ string getAddress();()
+ void setAddress(string address);()
+ list<Screen> getScreen();()
+ void setScreen(list<Screen> screen);()
+ list<Shows> getShows();()
+ void setShows(list<Shows>shows);()
+ City getCity();()
+ void setCity(City city);()
TheatreController
- map<City, list<Theatre>>city_theatre; : City
- allTheatre;list : Theatre
+ void addTheatre(Theatre theatre, City city);()
+ map<Theatre, list<Shows>> getAllShow(Movie movie, City city);()
-screen; list
Screen
- seats; list : Seat
- screenId : int
+ int getScreenId();()
+ void setScreenId(int screenId);()
+ list<Seat> getSeats();()
+ void setSeats(list<Seat> seats);()
-screen
Shows
- showId : int
- movie : Movie
- screen : Screen
- showStartTime : int
- bookedSeatIds;list : int
+ int getShowId();()
+ void setShowId(int showId);()
+ Movie getMovie();()
+ void setMovie(Movie movie);()
+ Screen getScreen();()
+ void setScreen(Screen screen);()
+ int getShowStartTime();()
+ void setShowStartTime(int showStartTime);()
+ list<int> getBookedSeatIds();()
+ void setBookedSeatIds(list<int> bookedSeatIds);()
Payment
- paymentId : int
+ void setPaymentID(int paymentId)()
+ int getPaymentID()()
-seats; list
Seat
- seatId : int
- row : int
- price : int
+ int getSeatId();()
+ void setSeatId(int seatId);()
+ int getRow();()
+ void setRow(int row);()
+ SeatCategory getSeatCategory();()
+ void setSeatCategory(SeatCategory seatCategory);()
Booking
- shows : Shows
- bookedSeats : Seat
- payment : Payment
+ void setShows(Shows shows);()
+ Shows getShows();()
+ list<Seat> getBookedSeats();()
+ void setBookedSeats(list<Seat> bookedSeats);()
+ Payment getPayment();()
+ void setPayment(Payment payment);()

Diagram: class diagram Page 1