Online movie booking system

Core capabilities:

* Barebone inventory and movie booking capability
* Integration with relevant payment services

# Functional requirements

1. Service should display cities where cinemas are located
2. Each theatre has multiple halls – 1 hall runs 1 show at a time
3. Service should display the movies released. Each movie will have multiple shows
4. User should be able to select 1 movie, service should display the cinemas running the movie
5. User selects a show at a particular cinema and book tickets
6. Service should be able to show the available seats in the hall. Users should be able to select multiple seats.
7. A user should be able to block seats for at-most 5 minutes before making a payment, after 5 minutes, session expires, and user must start all over again
8. Customer should be serviced in first come first served basis
9. Customers can cancel tickets and get refund

# Non-functional requirements

1. System should be highly concurrent. There will be multiple booking requests for the same seat at any point of time
2. Financial transactions should be secure and maintain ACID compliance in database

# Assumptions

1. Service doesn’t require user authentication
2. ~~Movie seats can be booked online only using this application. At the cinema counters, the front desk will login as guest to book tickets~~
3. ~~Only cash and card are the available payment options~~
4. Partial cancellation of tickets not allowed. In case of cancellation, all seats will be released
5. Cancellation time frame is 2 hours before a show time
6. A user cant book more than 10 tickets at a time

# ~~Use case diagram~~

~~There are 2 main actors in the system:~~

1. ~~Admin: Responsible for adding/deleting movies and their shows from the database.~~
2. ~~Guest: Search movies and book seats~~

~~Top use cases:~~

1. ~~Create/View/Cancel booking: to book a movie show ticket or view details about the show~~
2. ~~Make a payment for the booking~~
3. ~~Refund payment: on cancellation, refund of payment will be done if cancellation is done within allowed time frame (2 hours prior to show time)~~
4. ~~Customers can choose the seat numbers which are available for booking~~

# ~~Class diagram~~

~~Below are the main classes:~~

1. ~~Account: Admin will be able to add/remove movies/shows~~
2. ~~Guests: Can search and view movies and shows~~
3. ~~City: each city have multiple Cinemas~~
4. ~~Cinema: Theatre that runs movie shows~~
5. ~~CinemaHall: each cinema will have one or more halls~~
6. ~~Movie: have attributes such as title, description, release date, city name, etc.~~
7. ~~Show: Each movie will have multiple shows; each show will be played in a cinema hall~~
8. ~~CinemaHallSeat: each hall will have multiple seats~~
9. ~~Booking: booking against a movie show, have attributes such as booking id, movie id, show time, cinema id, number of seats, etc.~~
10. ~~Payment: payment done against a booking~~

# System APIs

The following are the API (REST/SOAP) definitions to:

1. Search movies

Search\_movies(account\_id, search\_string, city, start\_datetime, end\_datetime, results\_per\_page)

Parameters:

* Account\_id (String): To identify a registered account. This will be used to limit users to their allocated quota
* Search\_string (String): Keyword to search
* City (String): City to filter movies
* Start\_datetime (String): Filter movies with a starting date/time
* End\_datetime (String): Filter movies with ending date/time
* Results\_per\_page (Number): Number of rows returned per page. Max 30.

Returns:

* Array of movies and shows based on the search.

Sample JSON



1. Reserve seats

Reserve\_movies(account\_id, session\_id, movie\_id, show\_id, seats[])

Parameters:

* Account\_id (String): To identify a registered account
* Session\_id (Number): to track user session
* Movie\_id (Number): Movie to reserve
* Show\_id (Number): Show to reserve
* Seats (Number array): Seat numbers to reserve

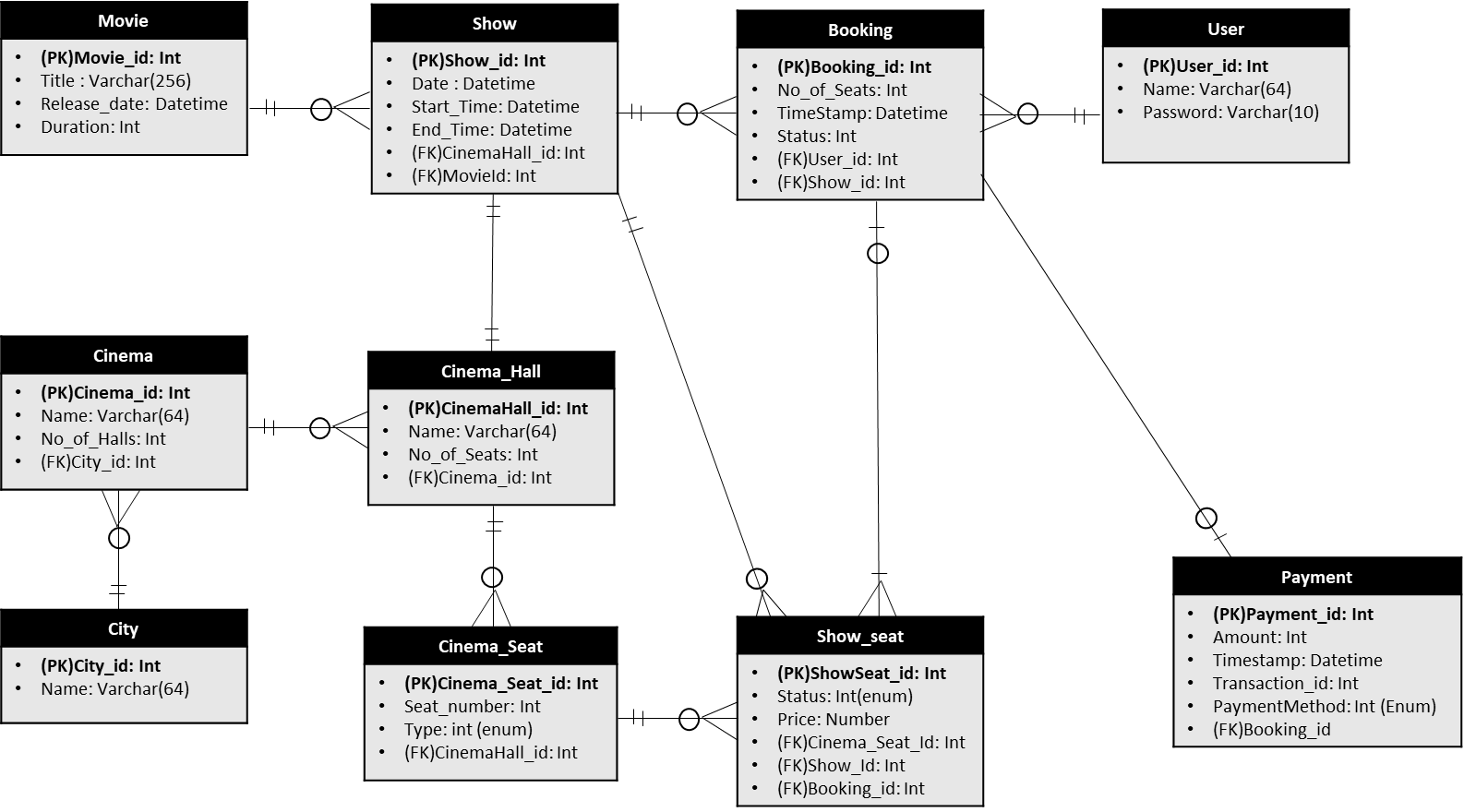
Returns:

* Returns the status of the reservation, which would be one of the following:
  + Successful
  + Failed – Show house full
  + Failed – Other users holding reserved seats

# Database Design

Database design is based on the following assumptions:

1. Each city will have multiple cinemas
2. Each cinema has multiple halls
3. Each movie has multiple shows, and each show has many bookings
4. One user can have more than 1 booking

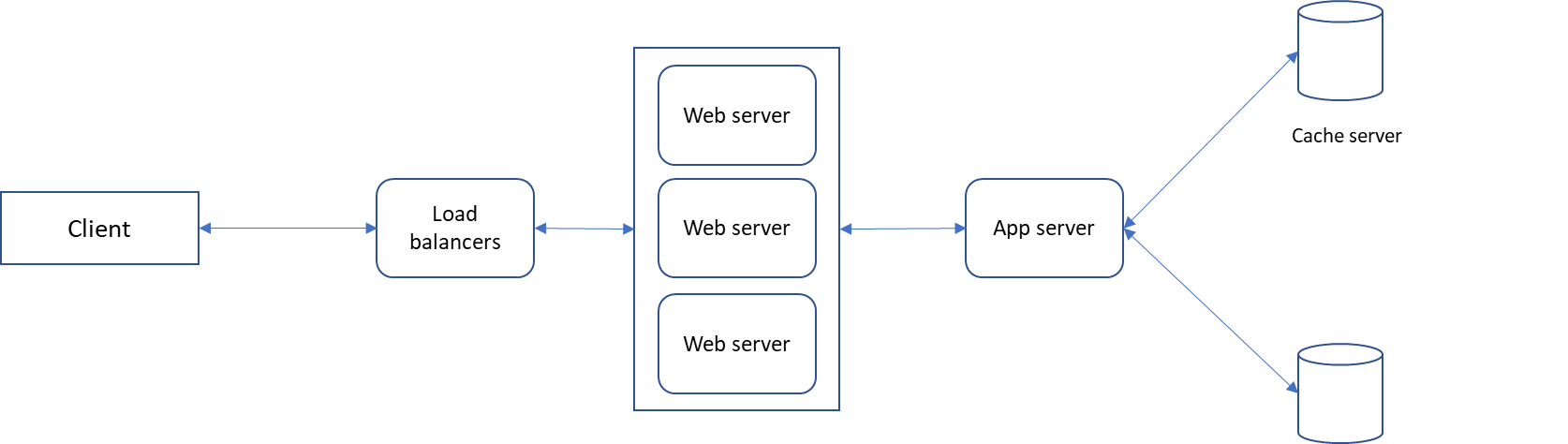


ERD diagram

# High-level Design

Components:

1. Webservers to manage user sessions
2. Application servers to manage ticket booking, storing data in database, working with cache servers to process reservations
3. Database to store booking, inventory and user data



# Detailed Component Design

**Ticket booking workflow.**



**Ticket Cancellation workflow:**



**Keeping track of active reservations**

2 daemon servers are needed to:

1. Keep a track of active reservations (ActiveReservationService)
2. Release connections for expired reservations (WaitingUserService)

ActiveReservationService

All reservations of a show can be stored in a Linked HashMap kind of data structure. Once a booking is complete, the system can remove the reservation from the HashMap. Since reservations can expire, head of the HashMap will point to the oldest reservation record. Once timeout is reached, reservation gets expired.

HashTable will store every reservations, where key = Show\_id, and value = Linked HashMap containing Booking\_id, creation timestamp.

In database, reservations are stored in the “Booking” table, expiry time is timestamp. Status field will have values such as: Reserved (1), Booked (2) and Expired (3). Once the booking is complete, status is updated from (1) to (2) and record is removed from the Linked HashMap. When a reservation is expired, update the status in table from (1) to (3) and remove from the HashMap. Periodically remove all records with status (3) from Booking table.

ActiveReservationService will call external payment services to process payments.

When a user session is expired or booking completed, WaitingUserService will be notified to serve any waiting users.

WaitingUserService

All waiting users of a show will be stored in another Linked HashMap. Remove any user from the HashMap when user cancels the request. Since it is first-come-first-served, the HashMap will always point to the longest waiting record.

A HashTable stores all waiting users for a show, where key = Show\_id, and value = Linked HashMap containing User\_id, start of wait time.

**Concurrency**

We need to maintain concurrency to make sure no 2 users can book the same seat. We can use SQL database to store transactions. We can use “Transaction Isolation Level” to lock rows before updating them (SQL Server)/Use commit after Update statements (Oracle). Below are sample codes:

SQL Server

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

BEGIN TRANSACTION;

-- Suppose we intend to reserve three seats (IDs: 54, 55, 56) for ShowID=99

Select \* From Show\_Seat where ShowID=99 && ShowSeatID in (54, 55, 56) && Status=0 -- free

-- if the number of rows returned by the above statement is three, we can update to

-- return success otherwise return failure to the user.

update Show\_Seat ...

update Booking ...

COMMIT TRANSACTION;

Oracle

BEGIN

-- Suppose we intend to reserve three seats (IDs: 54, 55, 56) for ShowID=99

Select \* From Show\_Seat where ShowID=99 && ShowSeatID in (54, 55, 56) && Status=0 -- free

-- if the number of rows returned by the above statement is three, we can update to

-- return success otherwise return failure to the user.

update Show\_Seat ...

update Booking ...

COMMIT;

END

# Fault Tolerance

Whenever the application server crashes, all active reservations can be read from the Database table “Booking”. We can also have a primary-secondary configuration for ActiveReservationService. It will also ensure when a primary crashes, the secondary server takes over.

For WaitingUserService we are not storing records in Database table. If we don’t have a secondary server for this service, we will not be able to recover the waiting users in case of a primary server failure.