Problem statement

You are given to build a movie ticket booking system. There are M x N seats which can be booked. The task is to implement a web application that implements the system with *blocking* and concurrency on the **seats (ticket)**. You can set a blocking timeout of 3 minutes on *reservation* till final confirmation after which the lock on the selected seat can be released.

Extrapolation:

The seats must belong to an **event** (an event which is a combination of a space ie. venue and time).

Further a UI is not being developed. I intend to test it using curl etc. Reason: I am not sure if a better exercise would be to wrap it up as APIs.

One ticket is generated per seat allocated or sold.

Functional specifications

User stories

User story: EVENT

(not implemented in code)

User creates an event in following steps.

- Step 1: Adds event details
    - Title
    - Brief
- Step 2: Pick a time and venue
    - Venue (selects from available set)
    - Time
- Step 3: Number of seats and prices
    - Seats
        - M rows
        - N columns
    - Pricing model
        - Standard price per seat
        - Additional price for particular seats
            - Rows / Columns / both
        - Discount on particular seats
            - Rows / Columns / both

User story: VENUE

(not implemented in code)

User creates a venue in the system

- Step 1: Create a venue
    - Name
    - Address
    - Seats
        - M rows
        - N columns

User story: BOOKING or TICKETS

Another user books one or more seats for the event.

- Step 1 : User finds the seats

        - User mentions the event
        - User mentions the preferred seats for suggestions. For example,
            - Number of seats
            - Preference by location of the seat
- Step 2 : User picks a suggested set of seats and blocks them
- Step 3 : User makes the payment and books the seats

Common User Stories: MISC

1. User signs up on the system
2. User logs in on the system

More to be added

System Modeling

Resources/Models:

1. User management
    a. Users
    b. Note: no admin APIs are being created
2. Booking system
    a. Events
    b. Venue
    c. Tickets

Actions/Resource handling:

1. User management
    a. Users
        i. Signup
        ii. Login
        iii. Logout
2. Book system
    a. Events
        i. Create
            1. Event Title
            2. Venue ID
            3. Time
            4. Standard Seat Pricing
            5. Waiting Seats
        ii. Delete
        iii. Get
        iv. Update
            1. Make Event Live
            2. Delete Event
    b. Venue
        i. Create
        ii. Delete
        iii. Get
        iv. Update
            1. Book Venue
            2. Free Venue
    c. Tickets (accessed by event creators only)
        i. Rules for ticket pricing given quality of seat
        ii. Reserve tickets/seats by age gender or gender
    d. Tickets (actions for normal users)
        i. Update request
            1. Block

2. Reserve
3. Cancel reservation

ii. Get or Query

1. Particular seat info
2. User can request N adjacent seats
3. User can request preference of seats towards the
    a. Center
    b. Back
    c. Corner - easier access to toilets
    d. Cost

Implementation Details

Mongodb Schema design

For booking management :-

1. Events schema
    a. Name - Text
    b. Brief or description - Text
    c. Venue schema reference - ID
    d. Start Time - Time
    e. End Time - Time
    f. Map of seat numbers with the Ticket ID
    g. Attendees (users) - Array of IDs
    h. Ticket Prices - Array of Integers
    i. isSoldOut - Boolean
    j. Created by (users) - ID
    k. Create on - Time
2. Venue
    a. Name - Text
    b. Address - Text
    c. PhoneNumber - Text
    d. bookedDates - sub-schema/struct
        i. Start Time
        ii. End Time
        iii. EventId
3. Ticket
    a. Event Id
    b. User id
    c. Seats - array of seat numbers
4. User
    a. Username
    b. Name
    c. Gender
    d. Age
    e. Upcoming event tickets
        i. EventID
        ii. Start date
        iii. End date
        iv. Price

For user management -

1. Account
    a. Username
    b. Email address
    c. Account type
        i. User id
        ii. Admin id

API design - version 0.1

1. Event
    a. POST /v0.1/event/
        i. name
        ii. description
        iii. venue
        iv. start_time
        v. end_time
        vi. std_ticket_price (optional)
        vii. waiting_seats
    b. PATCH /v.0/event/:id/?action=book&seats=[seat_numbers]
    c. GET /v.0/event/:id
        i. Not implemented
    d. GET /v.0/event/?filterBy=page&page=PAGE_NUM&items=NUM_PER_PAGE
        i. Returns all events
        ii. Not implemented
    e. DELETE /v.0/event/:id
        i. Deletes the event
        ii. This should also mandate refunding the users
        iii. Not implemented
2. Venue
    a. POST /v0.1/venue/
        i. name
        ii. description
        iii. address
        iv. seats
            1. rows
            2. columns
            3. max
    b. PATCH /v0.1/venue?action=book&event_id
        i. Books for a particular event (start & end dates)
    c. GET /v0.1/venue/

        i. Gets all venues
        ii. Accepts filterBy=FIELD
    d. GET /v0.1/venue/:id
        i. Gets information about a particular venue
    e. DELETE /v0.1/venue/:id
        i. Deletes venue from database
3. Tickets
    a. POST /v0.1/ticket/
        i. event
        ii. user
        iii. number of tickets
        iv. seating preferences
            1. arrangement
                a. adjacent
                b. equal groups
            2. factors
                a. number of females
                b. number of males
                c. number of senior citizens
    b. GET /v0.1/ticket/:id
        i. event
        ii. user
        iii. start time
        iv. end time
        v. price
    c. GET /v0.1/ticket/
        i. get all my tickets
    d. PATCH /v0.1/ticket/:id
        i. Not implemented
        ii. Could be used for postponing flights, etc
    e. DELETE /v0.1/ticket/:id
        i. Not implemented
4. User
    a. GET /v0.1/user/
        i. get current user info (minus the tickets)

Project files and directory structure

1. app.js - main file
2. routes_v0.1.js - routes for v0.1 apis
3. models/ - various schemas (self explainatory naming)
    a. event.js

    b. venue.js
    c. tickets.js
    d. user.js
4. controller/ - rest apis
    a. account.js - user management, login, logout, signup
    b. event.js - event related rest APIs
    c. venue.js - venue related rest APIs
    d. tickets.js - tickets related rest APIs
    e. user.js - user related rest APIs
5. services/
    a. booking/
        i. Micro-service for booking seats
    b. payment/
        i. Micro-service for payment

Other details:

1. Services are called from api's/front facing webapp via redis queue RPC
2. Services are implemented using architect prototype
3. No frontend is implemented for now