MERN 11 Project Part 7 - Payment and Generating Tickets

Creating a SingleMovie Page

Selecting Shows

Seating Arrangement

Make Payment and Get Tickets

## title:Creating a SingleMovie Page

When we click on any Movie on the HomePage A page that displays all the details about that Movie should open , We will call it SingleMovie component which will render the details of the particular movie that was clicked and also the Show timing of the movies

But before that you need to identify a Single Movie , we will be doing that wil the help of movie ID

So add this route in your movieRoutes files

movieRouter.get("/movie/:id", getMovieById);

Add this in the movieController

const getMovieById = async (req, res) => {

try {

const movie = await Movie.findById(req.params.id);

res.send({

success: true,

message: "Movie fetched successfully!",

data: movie,

});

} catch (err) {

res.send({

success: false,

message: err.message,

});

}

};

module.exports = { addMovie, getAllMovies, updateMovie, deleteMovie, getMovieById };

and write an Axios Call for this as well

In your api folder - Movie file add this Axios Instance

// Get a single movie by its id

export const getMovieById = async (id) => {

try {

const response = await axiosInstance.get(`/api/movies/movie/${id}`);

return response.data;

} catch (err) {

return err.response;

}

};

Now let;s build the Single Movie Component

SingleMovie.js under Home folder

import { useEffect, useState } from "react";

import { useNavigate, useParams } from "react-router-dom";

import { getMovieById } from "../../api/movie";

import { useDispatch } from "react-redux";

import { HideLoading, ShowLoading } from "../../redux/loaderSlice";

import { message, Input, Divider, Row, Col } from "antd";

import { CalendarOutlined } from "@ant-design/icons";

import moment from "moment";

import { getAllTheatresByMovie } from "../../api/shows";

const SingleMovie = () => {

const params = useParams();

const [movie, setMovie] = useState();

const [date, setDate] = useState(moment().format("YYYY-MM-DD"));

const [theatres, setTheatres] = useState([]);

const dispatch = useDispatch();

const navigate = useNavigate();

const handleDate = (e) => {

setDate(moment(e.target.value).format("YYYY-MM-DD"));

navigate(`/movie/${params.id}?date=${e.target.value}`);

};

const getData = async () => {

try {

dispatch(ShowLoading());

const response = await getMovieById(params.id);

if (response.success) {

setMovie(response.data);

} else {

message.error(response.message);

}

dispatch(HideLoading());

} catch (err) {

message.error(err.message);

dispatch(HideLoading());

}

};

const getAllTheatres = async () => {

try {

dispatch(ShowLoading());

const response = await getAllTheatresByMovie({ movie: params.id, date });

if (response.success) {

setTheatres(response.data);

} else {

message.error(response.message);

}

dispatch(HideLoading());

} catch (err) {

dispatch(HideLoading());

message.error(err.message);

}

};

useEffect(() => {

getData();

}, []);

useEffect(() => {

getAllTheatres();

}, [date]);

return (

<>

<div className="inner-container">

{movie && (

<div className="d-flex single-movie-div">

<div className="flex-Shrink-0 me-3 single-movie-img">

<img src={movie.poster} width={150} alt="Movie Poster" />

</div>

<div className="w-100">

<h1 className="mt-0">{movie.title}</h1>

<p className="movie-data">

Language: <span>{movie.language}</span>

</p>

<p className="movie-data">

Genre: <span>{movie.genre}</span>

</p>

<p className="movie-data">

Release Date:{" "}

<span>{moment(movie.date).format("MMM Do YYYY")}</span>

</p>

<p className="movie-data">

Duration: <span>{movie.duration} Minutes</span>

</p>

<hr />

<div className="d-flex flex-column-mob align-items-center mt-3">

<label className="me-3 flex-shrink-0">Choose the date:</label>

<Input

onChange={handleDate}

type="date"

min={moment().format("YYYY-MM-DD")}

className="max-width-300 mt-8px-mob"

value={date}

placeholder="default size"

prefix={<CalendarOutlined />}

/>

</div>

</div>

</div>

)}

{theatres.length === 0 && (

<div className="pt-3">

<h2 className="blue-clr">

Currently, no theatres available for this movie!

</h2>

</div>

)}

{theatres.length > 0 && (

<div className="theatre-wrapper mt-3 pt-3">

<h2>Theatres</h2>

{theatres.map((theatre) => {

return (

<div key={theatre.\_id}>

<Row gutter={24} key={theatre.\_id}>

<Col xs={{ span: 24 }} lg={{ span: 8 }}>

<h3>{theatre.name}</h3>

<p>{theatre.address}</p>

</Col>

<Col xs={{ span: 24 }} lg={{ span: 16 }}>

<ul className="show-ul">

{theatre.shows

.sort(

(a, b) =>

moment(a.time, "HH:mm") - moment(b.time, "HH:mm")

)

.map((singleShow) => {

return (

<li

key={singleShow.\_id}

onClick={() =>

navigate(`/book-show/${singleShow.\_id}`)

}

>

{moment(singleShow.time, "HH:mm").format(

"hh:mm A"

)}

</li>

);

})}

</ul>

</Col>

</Row>

<Divider />

</div>

);

})}

</div>

)}

</div>

</>

);

};

export default SingleMovie;

Note - Also add the Route for this component in App.js

<Route

path="/movie/:id"

element={

<ProtectedRoute>

<SingleMovie />

</ProtectedRoute>

}

/>

Add import -

import SingleMovie from "./pages/home/SingleMovie";

Need to fix this get-all-theatres-by-movie in showRoute.js

router.post("/get-all-theatres-by-movie", async (req, res) => {

try {

const { movie, date } = req.body;

const shows = await Show.find({ movie, date }).populate("theatre");

console.log("shows, ", shows);

// filter out the unique theatre

const uniqueTheatres = [];

shows.forEach((show) => {

const isTheatere = uniqueTheatres.find(

(theatre) => theatre.\_id === show.theatre.\_id

);

if (!isTheatere) {

const showsOfThisTheatre = shows.filter(

(showObj) => showObj.theatre.\_id === show.theatre.\_id

);

uniqueTheatres.push({

...show.theatre.\_doc,

shows: showsOfThisTheatre,

});

// uniqueTheatre -> [{theatre1, shows: [show1, show2]},{theatre2, shows:[show1, show2]}]

}

});

res.send({

success: true,

data: uniqueTheatres,

message: "All theatres by movie",

});

} catch (err) {

console.log(err);

res.send({

success: false,

message: "Failed to get all theatres by movie",

});

}

});

## title:Selecting Shows and Seating Arrangement

When we select a Show the Next thing that we need is the seating arrangement that should be visible to the user so they can go and book seats and then also make payment to get the tickets

home > BookShow.js

import { useEffect, useState } from "react";

import { useDispatch, useSelector } from "react-redux";

import { HideLoading, ShowLoading } from "../../redux/loaderSlice";

import { getShowById } from "../../api/shows";

import { useNavigate, useParams } from "react-router-dom";

import { message, Card, Row, Col, Button } from "antd";

import moment from "moment";

const BookShow = () => {

// Redux state and hooks

const { user } = useSelector((state) => state.users); // Extracting user from Redux state

const dispatch = useDispatch(); // Redux dispatch function

const [show, setShow] = useState(); // State for holding show details

const [selectedSeats, setSelectedSeats] = useState([]); // State for managing selected seats

const params = useParams(); // Extracting URL parameters

const navigate = useNavigate(); // Navigation hook

// Function to fetch show data by ID

const getData = async () => {

try {

dispatch(ShowLoading()); // Dispatching action to show loading state

const response = await getShowById({ showId: params.id }); // API call to fetch show details

if (response.success) {

setShow(response.data); // Setting state with fetched show data

// message.success(response.message); // Optional success message

console.log(response.data); // Logging show data to console

} else {

message.error(response.message); // Displaying error message if API call fails

}

dispatch(HideLoading()); // Dispatching action to hide loading state

} catch (err) {

message.error(err.message); // Handling errors from API call

dispatch(HideLoading()); // Hiding loading state on error

}

};

// Function to generate seat layout dynamically

const getSeats = () => {

let columns = 12; // Number of columns for seating arrangement

let totalSeats = 120; // Total number of seats

let rows = totalSeats / columns; // Calculating number of rows

return (

<div className="d-flex flex-column align-items-center">

<div className="w-100 max-width-600 mx-auto mb-25px">

<p className="text-center mb-10px">

Screen this side, you will be watching in this direction

</p>

<div className="screen-div">

{/\* Placeholder for screen display \*/}

</div>

</div>

<ul

className="seat-ul justify-content-center"

style={{ marginLeft: "25%" }}

>

{Array.from(Array(rows).keys()).map((row) =>

// Mapping rows

Array.from(Array(columns).keys()).map((column) => {

let seatNumber = row \* columns + column + 1; // Calculating seat number

let seatClass = "seat-btn"; // Default class for seat button

if (selectedSeats.includes(seatNumber)) {

seatClass += " selected"; // Adding 'selected' class if seat is selected

}

if (show.bookedSeats.includes(seatNumber)) {

seatClass += " booked"; // Adding 'booked' class if seat is already booked

}

if (seatNumber <= totalSeats) {

// Rendering seat button if seat number is valid

return (

<li key={seatNumber}>

{/\* Key added for React list rendering optimization \*/}

<button

className={seatClass}

onClick={() => {

// Function to handle seat selection/deselection

if (selectedSeats.includes(seatNumber)) {

setSelectedSeats(

selectedSeats.filter(

(curSeatNumber) => curSeatNumber !== seatNumber

)

);

} else {

setSelectedSeats([...selectedSeats, seatNumber]);

}

}}

>

{seatNumber}

</button>

</li>

);

}

})

)}

</ul>

<div className="d-flex bottom-card justify-content-between w-100 max-width-600 mx-auto mb-25px mt-3">

<div className="flex-1">

Selected Seats: <span>{selectedSeats.join(", ")}</span>

</div>

<div className="flex-shrink-0 ms-3">

Total Price:{" "}

<span>Rs. {selectedSeats.length \* show.ticketPrice}</span>

</div>

</div>

</div>

);

};

// Effect hook to fetch data on component mount

useEffect(() => {

getData();

}, []);

// JSX rendering

return (

<>

{show && (

<Row gutter={24}>

<Col span={24}>

<Card

title={

<div className="movie-title-details">

<h1>{show.movie.title}</h1>

<p>

Theatre: {show.theatre.name}, {show.theatre.address}

</p>

</div>

}

extra={

<div className="show-name py-3">

<h3>

<span>Show Name:</span> {show.name}

</h3>

<h3>

<span>Date & Time: </span>

{moment(show.date).format("MMM Do YYYY")} at{" "}

{moment(show.time, "HH:mm").format("hh:mm A")}

</h3>

<h3>

<span>Ticket Price:</span> Rs. {show.ticketPrice}/-

</h3>

<h3>

<span>Total Seats:</span> {show.totalSeats}

<span> &nbsp;|&nbsp; Available Seats:</span>{" "}

{show.totalSeats - show.bookedSeats.length}

</h3>

</div>

}

style={{ width: "100%" }}

>

{getSeats()} {/\* Rendering dynamic seat layout \*/}

</Card>

</Col>

</Row>

)}

</>

);

};

export default BookShow;

App.js

<Route

path="/book-show/:id"

element={

<ProtectedRoute>

<BookShow />

</ProtectedRoute>

}

/>

Explanation

**Rendering Seat Layout (getSeats function):**

Calculates the number of rows and columns based on totalSeats.

Maps through rows and columns to dynamically generate seat buttons.

Applies CSS classes (selected, booked) based on whether seats are selected or already booked (show.bookedSeats).

**Row and Column Calculation:**

The Array.from(Array(rows).keys()) creates an array of row indices (0 to rows-1).

The Array.from(Array(columns).keys()) creates an array of column indices (0 to columns-1).

Seat Number Calculation:

For each combination of row and column, seatNumber is calculated using the formula:

seatNumber = row \* columns + column + 1;

This formula ensures that each seat gets a unique identifier within the range from 1 to totalSeats.

Example Calculation:

Assuming columns = 12 and totalSeats = 120:

Rows (rows) would be totalSeats / columns = 10.

Iterating through each row (0 to 9) and each column (0 to 11), the seat number calculation progresses as follows:

For row = 0 and column = 0, seatNumber = 0 \* 12 + 0 + 1 = 1.

For row = 0 and column = 1, seatNumber = 0 \* 12 + 1 + 1 = 2.

…

For row = 1 and column = 0, seatNumber = 1 \* 12 + 0 + 1 = 13.

…

For row = 9 and column = 11, seatNumber = 9 \* 12 + 11 + 1 = 120.

## title:Book seats and Make Payment

Now the Next Step is to book tickets , for that we will use Stripe

What is Stripe

Stripe is an online payment processing platform that enables businesses to accept payments over the internet. It provides a suite of APIs (Application Programming Interfaces) that allows developers to integrate payment processing into websites and mobile applications easily. Here are some key aspects of Stripe:

Payment Methods: Stripe supports various payment methods including credit cards, debit cards, digital wallets (like Apple Pay and Google Pay), and bank transfers.

Integration: Developers can integrate Stripe into their applications using Stripe's APIs and SDKs (Software Development Kits). This integration allows businesses to securely capture and process payments without handling sensitive card information directly.

Security: Stripe is known for its robust security measures. It handles compliance and security requirements such as PCI-DSS (Payment Card Industry Data Security Standard) compliance, ensuring that payment information is handled securely.

Features: Apart from basic payment processing, Stripe offers features like subscription billing, recurring payments, invoicing, and customizable checkout experiences.

Dashboard and Reporting: Businesses using Stripe have access to a dashboard that provides insights into transactions, customer data, and other analytics related to payments.

Global Reach: Stripe supports payments in over 135 currencies and facilitates international transactions, making it suitable for businesses operating globally.

Overall, Stripe simplifies the process of online payment acceptance for businesses of all sizes, from startups to large enterprises, by providing a flexible and developer-friendly platform.

Stripe Website - <https://stripe.com/in>

Visit this website and log yourself in , after that make sure to generate your API keys - there will be two keys , one will be publishable key and another will be secret key , Generate both of them

In Stripe, the publishable key and secret key are both essential for securely integrating Stripe's payment services into your website or application:

**Publishable Key:**

The publishable key is used on the client-side (in the browser or mobile app) to identify your Stripe account when making API requests.

It is safe to expose this key in your frontend code (e.g., JavaScript), as it does not grant access to sensitive actions like issuing refunds or viewing transactions.

Its primary purpose is to initialize Stripe.js (Stripe's JavaScript library) on the client side and to generate secure tokens for handling payment details securely.

**Secret Key:**

The secret key, also known as the API key, is used on the server-side of your application to authenticate API requests to Stripe.

This key must be kept confidential and should never be exposed in frontend code or client-side applications.

It grants full access to your Stripe account, including the ability to perform actions like processing charges, creating refunds, managing subscriptions, and accessing sensitive data.

To summarize, the publishable key is used on the client-side for initiating transactions securely, while the secret key is used on the server-side for authenticating and performing operations securely with Stripe's API. It's crucial to handle these keys with care to maintain the security and integrity of your Stripe integration.

Here at the bottom left you will see two keys copy them and save them somewhere

Now install this npm library in your client folder **npm i react-stripe-checkout**

This will give you a checkout button and then will generate a payment component for card details

**How to add this in our code**

Go to BookShow.js Component and import React-stripe-checkout and then add that component like this at the bottom

import { useEffect, useState } from "react";

import { useDispatch, useSelector } from "react-redux";

import { HideLoading, ShowLoading } from "../../redux/loaderSlice";

import { getShowById } from "../../api/shows";

import { useNavigate, useParams } from "react-router-dom";

import { message, Card, Row, Col, Button } from "antd";

import moment from "moment";

import StripeCheckout from 'react-stripe-checkout'; // Stripe Checkout

useEffect(() => {

getData();

}, []);

const onToken=(token)=>{

console.log(token)

}

{getSeats()} {/\* Rendering dynamic seat layout \*/}

{selectedSeats.length > 0 && (

<StripeCheckout

token={onToken}

billingAddress

amount={selectedSeats.length \* show.ticketPrice \* 100}

stripeKey="pk\_test\_51PY5CiRteLJygSvEwi4Zd4R5ZKaKdY8VktV02VTgbK0KA7ATkaEpsK33AmpNQvIYJIZBnbpJuKUO3QG78eFkTH9B00Wyjxc4NL"

>

{/\* Use this one in some situation=> pk\_test\_eTH82XLklCU1LJBkr2cSDiGL001Bew71X8 \*/}

<div className="max-width-600 mx-auto">

<Button type="primary" shape="round" size="large" block>

Pay Now

</Button>

</div>

</StripeCheckout>

)}

**Discussion of the flow**

In our code snippet, you're using react-stripe-checkout to integrate Stripe for payments. Here’s an explanation of the key aspects:

**Stripe Checkout Integration:**

StripeCheckout component from react-stripe-checkout is used to handle the payment process. It takes several props, including token, billingAddress, amount, and stripeKey.

**Publishable Key:**

The stripeKey prop (pk\_test\_51JKPQWSJULHQ0FL7VOkMrOMFh0AHMoCFit29EgNlVRSvFkDxSoIuY771mqGczvd6bdTHU1EkhJpojOflzoIFGmj300Uj4ALqXa) is the publishable key obtained from our Stripe dashboard. This key is used on the client side to securely communicate with Stripe and generate a token representing the payment details.

**Generating Token:**

When a user clicks "Pay Now", StripeCheckout calls the token callback function (onToken) you provided. This function (onToken) receives a token object from Stripe containing payment details, which we will then use to process the payment on your server.

**Handling Payment:**

The token callback (onToken) logs the token object to the console in your example. Typically, we would send this token to our server (via an API call) along with other relevant information (like the total amount and selected items) to process the payment securely using Stripe’s API.

## title:Booking Model

This is the token that has been generated from the client side, this is not complete payment, we need to validate this token from our server side as well

For that we will create a Booking Model in our server side

Model -> bookingModel.js

const mongoose = require("mongoose");

const bookingSchema = new mongoose.Schema(

{

show: {

type: mongoose.Schema.Types.ObjectId,

ref: "shows",

},

user: {

type: mongoose.Schema.Types.ObjectId,

ref: "users",

},

seats: {

type: Array,

required: true,

},

transactionId: {

type: String,

required: true,

},

},

{ timestamps: true }

);

module.exports = mongoose.model("bookings", bookingSchema);

## title:Booking Routes

Create a file by the name bookingRoutes.js and make sure to attach it with your main server file

before that in your server side Install stripe

npm install stripe

also make sure you add your secret key in .env file

const router = require("express").Router();

const stripe = require("stripe")(process.env.STRIPE\_KEY);

const authMiddleware = require("../middlewares/authMiddleware");

const Booking = require("../models/bookingModel");

const Show = require("../models/showModel");

router.post("/make-payment", authMiddleware, async (req, res) => {

try {

const { token, amount } = req.body;

const customer = await stripe.customers.create({

email: token.email,

source: token.id,

});

const paymentIntent = await stripe.paymentIntents.create({

amount: amount,

currency: "usd",

customer: customer.id,

payment\_method\_types: ["card"],

receipt\_email: token.email,

description: "Token has been assigned to the movie!",

});

const transactionId = paymentIntent.id;

res.send({

success: true,

message: "Payment processing. You will receive a confirmation once the payment is complete",

data: transactionId,

});

} catch (err) {

res.send({

success: false,

message: err.message,

});

}

});

// Create a booking after the payment

router.post("/book-show", authMiddleware, async (req, res) => {

try {

const newBooking = new Booking(req.body);

await newBooking.save();

const show = await Show.findById(req.body.show).populate("movie");

const updatedBookedSeats = [...show.bookedSeats, ...req.body.seats];

await Show.findByIdAndUpdate(req.body.show, {

bookedSeats: updatedBookedSeats,

});

res.send({

success: true,

message: "New Booking done!",

data: newBooking,

});

} catch (err) {

res.send({

success: false,

message: err.message,

});

}

});

router.get("/get-all-bookings", authMiddleware, async (req, res) => {

try {

const bookings = await Booking.find({ user: req.body.userId })

.populate("user")

.populate("show")

.populate({

path: "show",

populate: {

path: "movie",

model: "movies",

},

})

.populate({

path: "show",

populate: {

path: "theatre",

model: "theatres",

},

});

res.send({

success: true,

message: "Bookings fetched!",

data: bookings,

});

} catch (err) {

res.send({

success: false,

message: err.message,

});

}

});

module.exports = router;

**EXPLANATION**

1. Route 1: POST /make-payment

This route handles the payment process using Stripe.

Middleware Used: authMiddleware is used to authenticate the user making the payment.

Functionality:

1. It expects token (containing payment information) and amount in the request body.
2. Creates a customer in Stripe using the provided email and payment source (token).
3. Creates a payment intent with the specified amount in USD, associating it with the customer.
4. Sends back a success response with the transaction ID if the payment is successful.
5. Sends back an error message if the payment fails.

2. Route 2: POST /book-show

This route creates a new booking after successful payment.

Middleware Used: authMiddleware ensures that the user is authenticated before booking a show.

Functionality:

1. Creates a new instance of Booking model using the data from the request body and saves it to the database.
2. Retrieves the show details associated with the booking (show) and updates the list of booked seats (bookedSeats) by adding the new booking's seats.
3. Sends back a success response with the newly created booking details if successful.
4. Sends back an error message if there's an issue with saving the booking or updating the show details.

## Create Axios Instances for all these Routes inside Booking.js file in your calls folder at the client side

booking.js

import { axiosInstance } from ".";

export const makePayment = async (token, amount) => {

try {

const response = await axiosInstance.post("/api/bookings/make-payment", {

token,

amount,

});

// console.log(token, amount, response);

return response.data;

} catch (err) {

return err.response;

}

};

export const bookShow = async (payload) => {

try {

const response = await axiosInstance.post(

"/api/bookings/book-show",

payload

);

console.log(response.data);

return response.data;

} catch (err) {

return err.response;

}

};

export const getAllBookings = async () => {

try {

const response = await axiosInstance.get("/api/bookings/get-all-bookings");

return response.data;

} catch (err) {

return err.response;

}

};

## title:Send Token from client and Make Payment

Now add these two functions in your BookingShow.js page in the client side , Here we will extend our onToken function and will also write a new function by the name book

import { useEffect, useState } from "react";

import { useDispatch, useSelector } from "react-redux";

import { HideLoading, ShowLoading } from "../../redux/loaderSlice";

import { getShowById } from "../../api/shows";

import { useNavigate, useParams } from "react-router-dom";

import { message, Card, Row, Col, Button } from "antd";

import moment from "moment";

import StripeCheckout from "react-stripe-checkout"; // Stripe Checkout

import { bookShow, makePayment } from "../../api/booking";

const BookShow = () => {

// Redux state and hooks

const { user } = useSelector((state) => state.users); // Extracting user from Redux state

const dispatch = useDispatch(); // Redux dispatch function

const [show, setShow] = useState(); // State for holding show details

const [selectedSeats, setSelectedSeats] = useState([]); // State for managing selected seats

const params = useParams(); // Extracting URL parameters

const navigate = useNavigate(); // Navigation hook

// Function to fetch show data by ID

const getData = async () => {

try {

dispatch(ShowLoading()); // Dispatching action to show loading state

const response = await getShowById({ showId: params.id }); // API call to fetch show details

if (response.success) {

setShow(response.data); // Setting state with fetched show data

// message.success(response.message); // Optional success message

console.log(response.data); // Logging show data to console

} else {

message.error(response.message); // Displaying error message if API call fails

}

dispatch(HideLoading()); // Dispatching action to hide loading state

} catch (err) {

message.error(err.message); // Handling errors from API call

dispatch(HideLoading()); // Hiding loading state on error

}

};

// Function to generate seat layout dynamically

const getSeats = () => {

let columns = 12;

let totalSeats = show.totalSeats;

let rows = Math.ceil(totalSeats / columns);

return (

<div className="d-flex flex-column align-items-center">

<div className="w-100 max-width-600 mx-auto mb-25px">

<p className="text-center mb-10px">

Screen this side, you will be watching in this direction

</p>

<div className="screen-div">

{/\* Placeholder for screen display \*/}

</div>

</div>

<ul

className="seat-ul justify-content-center"

style={{ marginLeft: "25%" }}

>

{Array.from(Array(rows).keys()).map((row) =>

// Mapping rows

Array.from(Array(columns).keys()).map((column) => {

let seatNumber = row \* columns + column + 1; // Calculating seat number

let seatClass = "seat-btn"; // Default class for seat button

if (selectedSeats.includes(seatNumber)) {

seatClass += " selected"; // Adding 'selected' class if seat is selected

}

if (show.bookedSeats.includes(seatNumber)) {

seatClass += " booked"; // Adding 'booked' class if seat is already booked

}

if (seatNumber <= totalSeats) {

// Rendering seat button if seat number is valid

return (

<li key={seatNumber}>

{/\* Key added for React list rendering optimization \*/}

<button

className={seatClass}

onClick={() => {

// Function to handle seat selection/deselection

if (selectedSeats.includes(seatNumber)) {

setSelectedSeats(

selectedSeats.filter(

(curSeatNumber) => curSeatNumber !== seatNumber

)

);

} else {

setSelectedSeats([...selectedSeats, seatNumber]);

}

}}

>

{seatNumber}

</button>

</li>

);

}

})

)}

</ul>

<div className="d-flex bottom-card justify-content-between w-100 max-width-600 mx-auto mb-25px mt-3">

<div className="flex-1">

Selected Seats: <span>{selectedSeats.join(", ")}</span>

</div>

<div className="flex-shrink-0 ms-3">

Total Price:{" "}

<span>Rs. {selectedSeats.length \* show.ticketPrice}</span>

</div>

</div>

</div>

);

};

// Effect hook to fetch data on component mount

useEffect(() => {

getData();

}, []);

const onToken = async (token) => {

try {

dispatch(ShowLoading());

const response = await makePayment(

token,

selectedSeats.length \* show.ticketPrice \* 100

);

if (response.success) {

message.success(response.message);

book(response.data);

console.log(response);

} else {

message.error(response.message);

}

dispatch(HideLoading());

} catch (err) {

message.error(err.message);

dispatch(HideLoading());

}

};

const book = async (transactionId) => {

try {

dispatch(ShowLoading());

const response = await bookShow({

show: params.id,

transactionId,

seats: selectedSeats,

user: user.\_id,

});

if (response.success) {

message.success("Show Booking done!");

navigate("/profile");

} else {

message.error(response.message);

}

dispatch(HideLoading());

} catch (err) {

message.error(err.message);

dispatch(HideLoading());

}

};

// JSX rendering

return (

<>

{show && (

<Row gutter={24}>

<Col span={24}>

<Card

title={

<div className="movie-title-details">

<h1>{show.movie.title}</h1>

<p>

Theatre: {show.theatre.name}, {show.theatre.address}

</p>

</div>

}

extra={

<div className="show-name py-3">

<h3>

<span>Show Name:</span> {show.name}

</h3>

<h3>

<span>Date & Time: </span>

{moment(show.date).format("MMM Do YYYY")} at{" "}

{moment(show.time, "HH:mm").format("hh:mm A")}

</h3>

<h3>

<span>Ticket Price:</span> Rs. {show.ticketPrice}/-

</h3>

<h3>

<span>Total Seats:</span> {show.totalSeats}

<span> &nbsp;|&nbsp; Available Seats:</span>{" "}

{show.totalSeats - show.bookedSeats.length}

</h3>

</div>

}

style={{ width: "100%" }}

>

{getSeats()} {/\* Rendering dynamic seat layout \*/}

{selectedSeats.length > 0 && (

<StripeCheckout

token={onToken}

billingAddress

amount={selectedSeats.length \* show.ticketPrice \* 100}

stripeKey="pk\_test\_51PY5CiRteLJygSvEwi4Zd4R5ZKaKdY8VktV02VTgbK0KA7ATkaEpsK33AmpNQvIYJIZBnbpJuKUO3QG78eFkTH9B00Wyjxc4NL"

>

{/\* Use this one in some situation=> pk\_test\_eTH82XLklCU1LJBkr2cSDiGL001Bew71X8 \*/}

<div className="max-width-600 mx-auto">

<Button type="primary" shape="round" size="large" block>

Pay Now

</Button>

</div>

</StripeCheckout>

)}

</Card>

</Col>

</Row>

)}

</>

);

};

export default BookShow;