Vehicle Parking Management System

1. Introduction

The Parking Management System is a web-based application designed to manage parking slots efficiently. It allows users to book parking slots, view available slots, and manage their bookings. The system also provides administrative functionalities to manage users, parking slots, and bookings. The application is built using Node.js, Express.js, MongoDB, and React.js.

2. Core Functionalities

2.1. User Authentication

Description:

User authentication is handled using JWT (JSON Web Tokens). Users can register, log in, and log out. The system uses secure cookies to store the JWT token and user data.

Backend Source Code:

Auth Route (authRoute.js):

```
import express from "express";

import bcrypt from "bcryptjs";

import jwt from "jsonwebtoken";

import { body, validationResult } from "express-validator";

import User from "../models/UserModel.js";

const router = express.Router();

const COOKIE_OPTIONS = {

httpOnly: true,

  secure: process.env.NODE_ENV === 'production',

sameSite: 'strict',

maxAge: 3600000 * 24 // 24 hours

};

// Login

router.post(`/signin`, [

body("email", "Enter a valid email!").isEmail(),

body("password", "Password can't be blank").notEmpty(),

  ], async (req, res) => {

const errors = validationResult(req);

    if (!errors.isEmpty()) {

      return res.status(400).json({ errors: errors.array() });
```

```
   }
const{ email, password } = req.body;


   try {
constexistingUser = await User.findOne({ email });
     if (!existingUser) {
       return res.status(404).json({ message: "User doesn't exist!" });
     }


constisPasswordOk = await bcrypt.compare(password, existingUser.password);
     if (!isPasswordOk) {
       return res.status(400).json({ message: "Invalid credentials!" });
     }
const token = jwt.sign(
{ id: existingUser._id, email: existingUser.email, role: existingUser.role },
process.env.JWT_SECRET,
{ expiresIn: "24h" }
     );
res.cookie("token", token, COOKIE_OPTIONS);
res.cookie("user", JSON.stringify(existingUser), COOKIE_OPTIONS);
     return res.status(200).json({ user: existingUser, token });
   } catch (err) {
console.error("Signin Error:", err);
     return res.status(500).json({ message: "Something went wrong!" });
   }
 }
);
// Register
router.post(`/signup`, async (req, res) => {
const{ firstName, lastName, contact, vehicle, email, password, confirmPassword } = req.body;


 if (!firstName || !lastName || !contact || !vehicle || !email || !password || password !== confirmPassword
|| password.length< 4) {
```

```
    return res.status(400).json({ message: "Invalid field!" });
  }

  try {
constexistingUser = await User.findOne({ email });
    if (existingUser) {
      return res.status(400).json({ message: "User already exists!" });
    }

consthashedPassword = await bcrypt.hash(password, 12);
constnewUser = await User.create({
      email,
      password: hashedPassword,
firstName,
lastName,
      vehicle,
      contact
    });

const token = jwt.sign(
{ id: newUser._id, email: newUser.email, role: newUser.role },
process.env.JWT_SECRET,
{ expiresIn: "24h" }
    );

res.cookie("token", token, COOKIE_OPTIONS);
res.cookie("user", JSON.stringify(newUser), COOKIE_OPTIONS);
res.status(200).json({ success: true, user: newUser, token, message: "Successfully Registered and Logged In!" });
  } catch (err) {
res.status(500).json({ message: "Something went wrong!" });
  }
});
```

```
// Logout
router.post("/logout", (req, res) => {
  try {
res.clearCookie("token", {
httpOnly: true,
     secure: process.env.NODE_ENV === "production",
sameSite: "strict",
   });
res.clearCookie("user", {
httpOnly: true,
     secure: process.env.NODE_ENV === "production",
sameSite: "strict",
   });

res.status(200).json({ status: "success", message: "Logged out successfully" });
  } catch (err) {
console.error("Logout error:", err);
res.status(500).json({ status: "error", message: "Internal server error" });
  }
});

export default router;
```

Frontend Source Code:

Sign-In Component (sign-in.jsx):

```
import { Input, Button, Typography } from "@material-tailwind/react";
import { Link, useNavigate } from "react-router-dom";
import { useState } from "react";
import { useDispatch } from "react-redux";
import { login } from "@/redux/apiCalls";

export function SignIn() {
const [email, setEmail] = useState("");
```

```
const [password, setPassword] = useState("");
const [error, setError] = useState("");
const dispatch = useDispatch();
const navigate = useNavigate();

consthandleClick = async (e) => {
e.preventDefault();
   try {
const response = await login(dispatch, { email, password });

     if (response?.user) {
       navigate("/dashboard/profile");
     } else {
setError(response?.error || "An error occurred. Please try again.");
     }
   } catch (error) {
setError("An error occurred. Please try again.");
   }
 };

 return (
<section className="m-8 flex gap-4">
<div className="w-full lg:w-3/5 mt-24">
<div className="text-center">
<Typography variant="h2" className="font-bold mb-4">Sign In</Typography>
<Typography variant="paragraph" color="blue-gray" className="text-lg font-normal">
          Enter your email and password to Sign In.
</Typography>
</div>
<form className="mt-8 mb-2 mx-auto w-80 max-w-screen-lglg:w-1/2">
<div className="mb-1 flex flex-col gap-6">
<Typography variant="small" color="blue-gray" className="-mb-3 font-medium">
           Your email
```

```
</Typography>
<Input
        size="lg"
        placeholder="name@mail.com"
        value={email}
onChange={(e) =>setEmail(e.target.value)}
className=" !border-t-blue-gray-200 focus:!border-t-gray-900"
labelProps={{
className: "before:content-noneafter:content-none",
        }}
       />
<Typography variant="small" color="blue-gray" className="-mb-3 font-medium">
        Password
</Typography>
<Input
        type="password"
        size="lg"
        placeholder="********"
        value={password}
onChange={(e) =>setPassword(e.target.value)}
className=" !border-t-blue-gray-200 focus:!border-t-gray-900"
labelProps={{
className: "before:content-noneafter:content-none",
        }}
       />
</div>

      {error && (
<Typography variant="small" color="red" className="mt-2 text-center">
        {error}
</Typography>
      )}
```

```
<Button className="mt-6" fullWidthonClick={handleClick}>
        Sign In
</Button>

<Typography variant="paragraph" className="text-center text-blue-gray-500 font-medium mt-4">
        Not registered?
<Link to="/auth/sign-up" className="text-gray-900 ml-1">Create account</Link>
</Typography>
</form>
</div>
<div className="w-2/5 h-full hidden lg:block">
<img
src="/img/pattern.png"
className="h-full w-full object-cover rounded-3xl"
      />
</div>
</section>
 );
}

export default SignIn;
```

Sign-Up Component (sign-up.jsx):

```
import { Card, Input, Checkbox, Button, Typography } from "@material-tailwind/react";
import { Link, useNavigate } from "react-router-dom";
import { useState } from "react";
import { useDispatch } from "react-redux";
import { login, signup } from "../../redux/apiCalls";

export function SignUp() {
const dispatch = useDispatch();
const navigate = useNavigate();
```

```
const [formData, setFormData] = useState({
firstName: "",
lastName: "",
   email: "",
   password: "",
confirmPassword: "",
   vehicle: "",
   contact: "",
 });

const [error, setError] = useState("");

consthandleChange = (e) => {
setFormData({ ...formData, [e.target.name]: e.target.value });
 };

consthandleSubmit = async (e) => {
e.preventDefault();
setError("");

   if (formData.password !== formData.confirmPassword) {
setError("Passwords do not match");
return;
   }
   try {
const response = await signup(dispatch, formData);
     if (response &&response.success) {
const{ email, password } = formData;
constloginResponse = await login(dispatch, { email, password });

       if (loginResponse&&loginResponse.success) {
         navigate('/dashboard/profile');
       } else {
```

```
setError("Automatic login failed. Please log in manually.");
          navigate('/auth/sign-in');
        }
      } else {
setError(response?.message || "Signup failed. Please try again.");
      }
    } catch (error) {
setError("Signup error, please try again.");
    }
  };

  return (
<section className="m-8 flex">
<div className="w-2/5 h-full hidden lg:block">
<imgsrc="/img/pattern.png" className="h-full w-full object-cover" alt="Pattern" />
</div>
<div className="w-full lg:w-3/5 flex flex-col items-center justify-center">
<div className="text-center">
<Typography variant="h2" className="font-bold mb-4">
          Join Us Today
</Typography>
<Typography variant="paragraph" color="blue-gray" className="text-lg font-normal">
          Enter your details to register.
</Typography>
</div>
<form className="mt-8 mb-2 mx-auto w-80 max-w-screen-lgsm:w-1/2" onSubmit={handleSubmit}>
<div className="flex flex-col space-y-4">
<Input size="lg" placeholder="First Name" name="firstName" value={formData.firstName}
onChange={handleChange} required />
<Input size="lg" placeholder="Last Name" name="lastName" value={formData.lastName}
onChange={handleChange} required />
<Input size="lg" type="email" placeholder="Email" name="email" value={formData.email}
onChange={handleChange} required />
```

```
<Input size="lg" type="password" placeholder="Password" name="password" value={formData.password} onChange={handleChange} required />

<Input size="lg" type="password" placeholder="Confirm Password" name="confirmPassword" value={formData.confirmPassword} onChange={handleChange} required />

<Input size="lg" placeholder="Vehicle No (Optional)" name="vehicle" value={formData.vehicle} onChange={handleChange} />

<Input size="lg" placeholder="Contact No (Optional)" name="contact" value={formData.contact} onChange={handleChange} />

</div>

      {error &&<Typography variant="small" className="text-red-500 mt-2">{error}</Typography>}

<Checkbox

        label={

<Typography variant="small" color="gray" className="flex items-center font-medium">

           I agree to the 

<a href="#" className="font-normal text-black transition-colors hover:text-gray-900 underline">

             Terms and Conditions

</a>

</Typography>

        }

        required

containerProps={{ className: "-ml-3" }}

      />

<Button type="submit" className="mt-4" fullWidth>

        Sign Up

</Button>

<Typography variant="paragraph" className="text-center text-blue-gray-500 font-medium mt-4">

        Already have an account?

<Link to="/auth/sign-in" className="text-gray-900 ml-1 font-semibold underline">

          Sign In

</Link>
```

```
</Typography>
</form>
</div>
</section>
 );
}

export default SignUp;
```

2.2. Parking Slot Management

Description:

Admins can create, update, and delete parking slots. Users can view available parking slots and book them. The system ensures that only authorized users (admins or moderators) can modify parking slots.

Backend Source Code:

Parking Controller (parkingController.js):

```
import Parking from "../models/ParkingModel.js";

// Create a parking slot (Admin only)
export constcreateParkingSlot = async (req, res) => {
  try {
const{ slotNumber, location, price } = req.body;
constnewSlot = new Parking({ slotNumber, location, price });
    await newSlot.save();
res.status(201).json(newSlot);
  } catch (error) {
res.status(500).json({ message: "Error creating parking slot", error });
  }
};

// Update a parking slot (Admin or Moderator)
export constupdateParkingSlot = async (req, res) => {
  try {
const{ id } = req.params;
```

```
const updates = req.body;
constupdatedSlot = await Parking.findByIdAndUpdate(id, updates, { new: true });
res.json(updatedSlot);
   } catch (error) {
res.status(500).json({ message: "Error updating parking slot", error });
   }
};

// Delete a parking slot (Admin only)
export constdeleteParkingSlot = async (req, res) => {
   try {
const{ id } = req.params;
      await Parking.findByIdAndDelete(id);
res.status(204).send();
   } catch (error) {
res.status(500).json({ message: "Error deleting parking slot", error });
   }
};

// Get all parking slots (All users)
export constgetAllParkingSlots = async (req, res) => {
   try {
const slots = await Parking.find();
res.json(slots);
   } catch (error) {
res.status(500).json({ message: "Error fetching parking slots", error });
   }
};

// Get available parking slots (All users)
export constgetAvailableParkingSlots = async (req, res) => {
   try {
constavailableSlots = await Parking.find({ status: "Available" });
```

```
res.json(availableSlots);
   } catch (error) {
res.status(500).json({ message: "Error fetching available slots", error });
   }
};

// Book a parking slot (All users)
export constbookParkingSlot = async (req, res) => {
const{ id } = req.params;
const{ vehicleNumber, fromDate, toDate } = req.body;
constuserId = req.user._id;

   try {
constparkingSlot = await Parking.findById(id);
      if (!parkingSlot) {
         return res.status(404).json({ message: "Parking slot not found" });
      }

constoverlappingBooking = parkingSlot.bookings.some(
         (booking) => (fromDate<= booking.toDate&&toDate>= booking.fromDate)
      );

      if (overlappingBooking) {
         return res.status(400).json({ message: "Slot already booked for the selected date range" });
      }

parkingSlot.bookings.push({ vehicleNumber, fromDate, toDate });
parkingSlot.status = "Occupied";
parkingSlot.bookedBy = userId;
      await parkingSlot.save();

res.status(200).json(parkingSlot);
   } catch (error) {
```

```
res.status(500).json({ message: "Error booking parking slot", error });
  }
};
```

Frontend Source Code:

Parking Management Component (parkingManagement.jsx):

```
import React, { useState, useEffect } from "react";
import {
 Card,
CardHeader,
CardBody,
 Typography,
 Chip,
 Button,
 Dialog,
DialogHeader,
DialogBody,
DialogFooter,
 Input,
} from "@material-tailwind/react";
import { PlusIcon, PencilIcon, TrashIcon, BookmarkIcon } from "@heroicons/react/24/outline";
import { publicRequest } from "@/requestMethods";
import DatePicker from "react-datepicker";
import "react-datepicker/dist/react-datepicker.css";

constParkingManagement = () => {
const [parkingSlots, setParkingSlots] = useState([]);
const [bookings, setBookings] = useState([]);
const [openDialog, setOpenDialog] = useState(false);
const [openEditDialog, setOpenEditDialog] = useState(false);
const [openBookingDialog, setOpenBookingDialog] = useState(false);
const [newSlot, setNewSlot] = useState({ slotNumber: "", location: "", price: 0 });
const [editSlot, setEditSlot] = useState({ _id: "", slotNumber: "", location: "", price: 0 });
```

```
const [bookingSlot, setBookingSlot] = useState({ _id: "", vehicleNumber: "", fromDate: new Date(), toDate: new Date() });

const [error, setError] = useState("");

const [currentPage, setCurrentPage] = useState(1);

const [itemsPerPage] = useState(5);

const [userRole, setUserRole] = useState("");


useEffect(() => {

constfetchParkingSlots = async () => {

    try {

const token = localStorage.getItem("token");

      if (!token) {

console.error("No token found. Please log in.");

return;

      }


const response = await publicRequest.get("/parking/slots", {

        headers: {

          Authorization: `Bearer ${token}`,

        },

      });

setParkingSlots(response.data);

    } catch (error) {

console.error("Error fetching parking slots:", error);

    }

  };


fetchParkingSlots();

 }, []);


consthandleSubmit = async (e) => {

e.preventDefault();

   try {
```

```
const{ slotNumber, location, price } = newSlot;

    if (!slotNumber || !location || price === undefined) {
setError("Slot number, location, and price are required.");
return;
    }

constslotToCreate = {
slotNumber,
      location,
      price,
      status: "Available",
    };

const token = localStorage.getItem("token");
const response = await publicRequest.post("/parking/slots", slotToCreate, {
      headers: {
        Authorization: `Bearer ${token}`,
      },
    });

setParkingSlots([...parkingSlots, response.data]);
setOpenDialog(false);
setNewSlot({ slotNumber: "", location: "", price: 0 });
setError("");
   } catch (error) {
console.error("Error creating parking slot:", error);
setError("An error occurred while creating the parking slot.");
   }
 };

 return (
<>
```

```
<div className="mt-12 mb-8 flex flex-col gap-12">
<Card>
<CardHeader variant="gradient" color="gray" className="mb-8 p-6 flex justify-between items-center">
<Typography variant="h6" color="white">
          Parking Slots Management
</Typography>
        {userRole == "admin" && (
<Button
color="white"
className="flex items-center gap-2"
onClick={() =>setOpenDialog(true)}
>
<PlusIconclassName="h-5 w-5" />
            Add New Slot
</Button>
        )}
</CardHeader>
<CardBodyclassName="overflow-x-scroll px-0 pt-0 pb-2">
        {parkingSlots.length === 0 ? (
<Typography variant="h6" color="blue-gray" className="text-center py-8">
            No data available
</Typography>
        ) : (
<>
<table className="w-full min-w-[640px] table-auto">
<thead>
<tr>
                {["Slot Number", "Location", "Status", "Price", "Vehicle Number", "Booked By", "Actions"].map((el)
=> (
<th
                    key={el}
className="border-b border-blue-gray-50 py-3 px-5 text-left"
>
```

```
<Typography
                    variant="small"
className="text-[11px] font-bold uppercase text-blue-gray-400"
>
                    {el}
</Typography>
</th>
                ))}
</tr>
</thead>
<tbody>
              {parkingSlots.map((slot) => (
<tr key={slot._id}>
<td className="py-3 px-5 border-b border-blue-gray-50">
<Typography className="text-xs font-semibold text-blue-gray-600">
                    {slot.slotNumber}
</Typography>
</td>
<td className="py-3 px-5 border-b border-blue-gray-50">
<Typography className="text-xs font-semibold text-blue-gray-600">
                    {slot.location}
</Typography>
</td>
<td className="py-3 px-5 border-b border-blue-gray-50">
<Chip
                    variant="gradient"
color={slot.status === "Available" ? "green" : "red"}
                    value={slot.status}
className="py-0.5 px-2 text-[11px] font-medium w-fit"
                  />
</td>
<td className="py-3 px-5 border-b border-blue-gray-50">
<Typography className="text-xs font-semibold text-blue-gray-600">
```

```
&#8377; {slot.price}
</Typography>
</td>
<td className="py-3 px-5 border-b border-blue-gray-50">
<Typography className="text-xs font-semibold text-blue-gray-600">
                    {slot.vehicleNumber || "N/A"}
</Typography>
</td>
<td className="py-3 px-5 border-b border-blue-gray-50">
<Typography className="text-xs font-semibold text-blue-gray-600">
                    {slot.bookedBy?.username || "N/A"}
</Typography>
</td>
<td className="py-3 px-5 border-b border-blue-gray-50">
                  {userRole === "user" ? (
<Button className=" flex flex-row "
color="blue"
                    size="sm"
onClick={() => {
setBookingSlot({ _id: slot._id, vehicleNumber: "", fromDate: new Date(), toDate: new Date() });
setOpenBookingDialog(true);
                    }}
>
<BookmarkIconclassName="h-4 w-4" />
                    Book
</Button>
                  ) : (
<>
<Button
color="blue"
                     size="sm"
className="mr-2"
onClick={() => {
```

```
setEditSlot(slot);
setOpenEditDialog(true);
                        }}
>
<PencilIconclassName="h-4 w-4" />
</Button>
                    {userRole == "admin" && (<Button
color="red"
                        size="sm"
onClick={() =>handleDelete(slot._id)}
>
<TrashIconclassName="h-4 w-4" />
</Button>
                    )}
</>
                  )}
</td>
</tr>
              ))}
</tbody>
</table>
<div className="flex justify-center mt-4">
          {Array.from({ length: Math.ceil(parkingSlots.length / itemsPerPage) }, (_, i) => (
<Button
              key={i + 1}
color={currentPage === i + 1 ? "blue" : "gray"}
onClick={() => paginate(i + 1)}
className="mx-1"
>
              {i + 1}
</Button>
          ))}
</div>
```

```
</>
        )}
</CardBody>
</Card>
</div>

<Dialog open={openDialog} handler={() =>setOpenDialog(!openDialog)}>
<DialogHeader>Add New Parking Slot</DialogHeader>
<DialogBody>
<form onSubmit={handleSubmit}>
<div className="space-y-4">
<Input
            label="Slot Number"
            name="slotNumber"
            value={newSlot.slotNumber}
onChange={handleInputChange}
            required
          />
<Input
            label="Location"
            name="location"
            value={newSlot.location}
onChange={handleInputChange}
            required
          />
<Input
            label="Price"
            name="price"
            type="number"
            value={newSlot.price}
onChange={handleInputChange}
            min="0"
            required
```

```
        />
</div>
</form>
</DialogBody>
<DialogFooter>
<Button
        variant="text"
color="red"
onClick={() =>setOpenDialog(false)}
className="mr-2"
>
        Cancel
</Button>
<Button variant="gradient" color="green" onClick={handleSubmit}>
        Add Slot
</Button>
</DialogFooter>
</Dialog>
</>
  );
};

export default ParkingManagement;
```

2.3. Booking Management

Description:

Users can create, view, and delete their bookings. Admins and moderators can view all bookings. The system ensures that bookings are linked to both users and parking slots.

Backend Source Code:

Booking Controller (bookingController.js):

```
import Booking from "../models/BookingModel.js";
import Parking from "../models/ParkingModel.js";
```

```
// Create a new booking
export constcreateBooking = async (req, res) => {
   try {
const{ parkingSlot, vehicleNumber, fromDate, toDate } = req.body;
constuserId = req.user.id;

const parking = await Parking.findById(parkingSlot);
      if (!parking) {
         return res.status(404).json({ message: "Parking slot not found" });
      }

const booking = new Booking({
parkingSlot,
bookedBy: userId,
vehicleNumber,
fromDate,
toDate,
      });

      await booking.save();
res.status(201).json(booking);
   } catch (error) {
res.status(500).json({ message: "Error creating booking", error });
   }
};

// Get all bookings for a user
export constgetUserBookings = async (req, res) => {
   try {
constuserId = req.params.userId;
const bookings = await Booking.find({ bookedBy: userId })
.populate("parkingSlot")
```

```
.populate("bookedBy", "firstName email");
res.status(200).json(bookings);
   } catch (error) {
res.status(500).json({ message: "Error fetching bookings", error });
   }
};

// Get all bookings (for admins/moderators)
export constgetAllBookings = async (req, res) => {
   try {
const bookings = await Booking.find()
.populate("parkingSlot")
.populate("bookedBy", "firstName email");
res.status(200).json(bookings);
   } catch (error) {
res.status(500).json({ message: "Error fetching all bookings", error });
   }
};

// Delete a booking
export constdeleteBooking = async (req, res) => {
const{ bookingId } = req.params;
   try {
const booking = await Booking.findByIdAndDelete(bookingId);
      if (!booking) {
         return res.status(404).json({ message: "Booking not found" });
      }

constparkingSlot = await Parking.findById(booking.parkingSlot);
      if (parkingSlot) {
parkingSlot.status = "Available";
parkingSlot.bookedBy = null;
         await parkingSlot.save();
```

```
        }

res.status(200).json({ message: "Booking deleted successfully" });
    } catch (error) {
res.status(500).json({ message: "Error deleting booking", error });
    }
};
```

Frontend Source Code:

Booking Management Component (bookingManagement.jsx):

```
import React, { useState, useEffect } from "react";
import {
  Card,
CardHeader,
CardBody,
  Typography,
  Chip,
  Button,
  Dialog,
DialogHeader,
DialogBody,
DialogFooter,
  Input,
} from "@material-tailwind/react";
import { publicRequest } from "@/requestMethods";

constBookingManagement = () => {
const [bookings, setBookings] = useState([]);
const [error, setError] = useState("");
const [currentPage, setCurrentPage] = useState(1);
const [itemsPerPage] = useState(5);
const [userRole, setUserRole] = useState("");
const [userId, setUserId] = useState("");
```

```
useEffect(() => {
constfetchBookings = async () => {
    try {
const token = localStorage.getItem("token");
      if (!token) {
console.error("No token found. Please log in.");
return;
      }

      let url = `/bookings/user/${userId}`;
      if (userRole === "admin" || userRole === "moderator") {
url = "/bookings";
      }

const response = await publicRequest.get(url, {
        headers: {
          Authorization: `Bearer ${token}`,
        },
      });

      if (Array.isArray(response.data)) {
setBookings(response.data);
      } else {
setBookings([]);
      }
    } catch (error) {
setError("An error occurred while fetching bookings.");
setBookings([]);
    }
  };

fetchBookings();
```

```
  }, [userId, userRole]);


constindexOfLastItem = currentPage * itemsPerPage;
constindexOfFirstItem = indexOfLastItem - itemsPerPage;
constcurrentItems = bookings.slice(indexOfFirstItem, indexOfLastItem);


const paginate = (pageNumber) =>setCurrentPage(pageNumber);


  return (
<>
<div className="mt-12 mb-8 flex flex-col gap-12">
<Card>
<CardHeader variant="gradient" color="gray" className="mb-8 p-6 flex justify-between items-center">
<Typography variant="h6" color="white">
            Booking Management
</Typography>
</CardHeader>
<CardBodyclassName="overflow-x-scroll px-0 pt-0 pb-2">
{!bookings || bookings.length === 0 ? (
<Typography variant="h6" color="blue-gray" className="text-center py-8">
              No bookings available
</Typography>
          ) : (
<>
<table className="w-full min-w-[640px] table-auto">
<thead>
<tr>
                  {["Slot Number", "Location", "Vehicle Number", "From Date", "To Date", "Booked By"].map((el) =>
(
<th
                      key={el}
className="border-b border-blue-gray-50 py-3 px-5 text-left"
>
```

```
<Typography
                    variant="small"
className="text-[11px] font-bold uppercase text-blue-gray-400"
>
                    {el}
</Typography>
</th>
                ))}
</tr>
</thead>
<tbody>
              {bookings.map((booking) => (
<tr key={booking._id}>
<td className="py-3 px-5 border-b border-blue-gray-50">
<Typography className="text-xs font-semibold text-blue-gray-600">
                    {booking.parkingSlot?.slotNumber || "N/A"}
</Typography>
</td>
<td className="py-3 px-5 border-b border-blue-gray-50">
<Typography className="text-xs font-semibold text-blue-gray-600">
                    {booking.parkingSlot?.location || "N/A"}
</Typography>
</td>
<td className="py-3 px-5 border-b border-blue-gray-50">
<Typography className="text-xs font-semibold text-blue-gray-600">
                    {booking.vehicleNumber || "N/A"}
</Typography>
</td>
<td className="py-3 px-5 border-b border-blue-gray-50">
<Typography className="text-xs font-semibold text-blue-gray-600">
                    {new Date(booking.fromDate).toLocaleDateString()}
</Typography>
</td>
```

```
<td className="py-3 px-5 border-b border-blue-gray-50">
<Typography className="text-xs font-semibold text-blue-gray-600">
                    {new Date(booking.toDate).toLocaleDateString()}
</Typography>
</td>
<td className="py-3 px-5 border-b border-blue-gray-50">
<Typography className="text-xs font-semibold text-blue-gray-600">
                    {booking.bookedBy?.firstName  || "N/A"}
</Typography>
</td>
</tr>
                ))}
</tbody>
</table>
<div className="flex justify-center mt-4">
              {Array.from({ length: Math.ceil(bookings.length / itemsPerPage) }, (_, i) => (
<Button
                  key={i + 1}
color={currentPage === i + 1 ? "blue" : "gray"}
onClick={() => paginate(i + 1)}
className="mx-1"
>
                  {i + 1}
</Button>
              ))}
</div>
</>
          )}
</CardBody>
</Card>
</div>

    {error && (
```

```
<Typography color="red" className="text-center mt-4">
        {error}
</Typography>
    )}
</>
  );
};

export default BookingManagement;
```

2.4. User Management

Description:

Admins can view, update, and delete users. Admins can also update user roles (User, Moderator, Admin).

Backend Source Code:

User Controller (userController.js):

```
import User from "../models/UserModel.js";

// Get User
export constgetUser = async (req, res) => {
  try {
const user = await User.findById(req.params.id);
res.status(201).json(user);
  } catch (err) {
res.status(500).json(err);
  }
};

// Get all Users (with pagination)
export constgetAllUser = async (req, res) => {
  try {
const page = parseInt(req.query.page) || 1;
const limit = parseInt(req.query.limit) || 10;
```

```
const skip = (page - 1) * limit;

const users = await User.find().skip(skip).limit(limit).exec();
consttotalUsers = await User.countDocuments();
consttotalPages = Math.ceil(totalUsers / limit);

res.status(200).json({ users, totalPages, currentPage: page });
 } catch (err) {
res.status(500).json({ message: "Server error", error: err.message });
 }
};

// Update User
export constupdateUser = async (req, res) => {
 try {
const{ id } = req.params;
const{ firstName, lastName, contact, vehicle } = req.body;

constupdatedUser = await User.findByIdAndUpdate(
    id,
{ firstName, lastName, contact, vehicle },
{ new: true }
  );

  if (!updatedUser) {
    return res.status(404).json({ error: "User not found." });
  }

res.status(200).json(updatedUser);
 } catch (err) {
res.status(500).json({ error: "Failed to update user." });
 }
};
```

```
// Update User Role (Admin only)
export constupdateUserRole = async (req, res) => {
  try {
const{ id } = req.params;
const{ role } = req.body;

    if (req.user._id === id &&req.user.role === 'admin') {
      return res.status(403).json({ error: "Admins cannot change their own role." });
    }

constupdatedUser = await User.findByIdAndUpdate(
      id,
{ role },
{ new: true }
    );

    if (!updatedUser) {
      return res.status(404).json({ message: "User not found" });
    }

res.status(200).json(updatedUser);
  } catch (err) {
res.status(500).json({ message: "Server error", error: err.message });
  }
};

// Delete User (Admin only)
export constdeleteUser = async (req, res) => {
  try {
    await User.findByIdAndDelete(req.params.id);
res.status(200).json("User has been deleted...");
  } catch (err) {
```

```
res.status(500).json(err);
  }
};
```

Frontend Source Code:

User Management Component (userSection.jsx):

```
import React, { useState, useEffect } from "react";
import {
  Card,
CardHeader,
CardBody,
  Typography,
  Avatar,
  Chip,
  Button,
  Select,
  Option,
  Input,
} from "@material-tailwind/react";
import { publicRequest } from "@/requestMethods";
import { UserModal } from "@/components/UserModal";

export function UserSection() {
const [users, setUsers] = useState([]);
const [loading, setLoading] = useState(true);
const [currentPage, setCurrentPage] = useState(1);
const [totalPages, setTotalPages] = useState(1);
const [selectedUser, setSelectedUser] = useState(null);
const [isModalOpen, setIsModalOpen] = useState(false);
const [editingUserId, setEditingUserId] = useState(null);
const [editedUserDetails, setEditedUserDetails] = useState({});
const [currentUserRole, setCurrentUserRole] = useState("");
```

```
useEffect(() => {
constfetchUsers = async () => {
    try {
const token = localStorage.getItem("token");
      if (!token) {
        throw new Error("No token found. Please log in.");
      }

const res = await publicRequest.get(`/user?page=${currentPage}&limit=10`, {
        headers: {
          Authorization: `Bearer ${token}`,
        },
      });
setUsers(res.data.users);
setTotalPages(res.data.totalPages);
    } catch (err) {
console.error("Error fetching users:", err);
alert(err.response?.data?.error || "Failed to fetch users. Please log in.");
    } finally {
setLoading(false);
    }
  };

fetchUsers();
 }, [currentPage]);

consthandlePageChange = (newPage) => {
  if (newPage>= 1 &&newPage<= totalPages) {
setCurrentPage(newPage);
  }
 };

consthandleRoleChange = async (userId, newRole) => {
```

```
   try {
const token = localStorage.getItem("token");
     if (!token) {
       throw new Error("No token found. Please log in.");
     }

const res = await publicRequest.put(
       `/user/${userId}/role`,
{ role: newRole },
       {
         headers: {
           Authorization: `Bearer ${token}`,
         },
       }
     );
constupdatedUser = res.data;

setUsers((prevUsers) =>
prevUsers.map((user) =>
user._id === updatedUser._id ? updatedUser : user
       )
     );
   } catch (err) {
console.error("Error updating user role:", err.response?.data?.error || err.message);
alert(err.response?.data?.error || "Failed to update user role.");
   }
 };

  return (
<div className="mt-12 mb-8 flex flex-col gap-12">
<Card>
<CardHeader variant="gradient" color="gray" className="mb-8 p-6">
<Typography variant="h6" color="white">
```

```
        Authors Table
</Typography>
</CardHeader>
<CardBodyclassName="overflow-x-scroll px-0 pt-0 pb-2">
<table className="w-full min-w-[640px] table-auto">
<thead>
<tr>
          {["Author", "Contact", "Vehicle", "Role", "Status", "Actions"].map((el) => (
<th
            key={el}
className="border-b border-blue-gray-50 py-3 px-5 text-left"
>
<Typography
              variant="small"
className="text-[11px] font-bold uppercase text-blue-gray-400"
>
              {el}
</Typography>
</th>
          ))}
</tr>
</thead>
<tbody>
        {users.map(({ _id, profilePicture, firstName, lastName, email, role, verified, contact, vehicle }) => {
constfullName = `${firstName} ${lastName}`;
constisEditing = editingUserId === _id;
constcanEdit = canEditUser(role, _id);

          return (
<tr key={_id}>
<td className="py-3 px-5 border-b border-blue-gray-50">
<div className="flex items-center gap-4">
<Avatar src={profilePicture || "/img/bruce-mars.jpeg"} alt={fullName} size="sm" variant="rounded" />
```

```
<div>
                    {isEditing ? (
<>
<Input
                        value={editedUserDetails.firstName}
onChange={(e) =>handleInputChange(e, "firstName")}
                        placeholder="First Name"
className="mb-2"
                       />
<Input
                        value={editedUserDetails.lastName}
onChange={(e) =>handleInputChange(e, "lastName")}
                        placeholder="Last Name"
                       />
</>
                    ) : (
<>
<Typography variant="small" color="blue-gray" className="font-semibold">
                        {fullName}
</Typography>
<Typography className="text-xs font-normal text-blue-gray-500">
                        {email}
</Typography>
</>
                    )}
</div>
</div>
</td>
<td className="py-3 px-5 border-b border-blue-gray-50">
                {isEditing ? (
<Input
                    value={editedUserDetails.contact}
onChange={(e) =>handleInputChange(e, "contact")}
```

```
                    placeholder="Contact"
                  />
                ) : (
<Typography className="text-xs font-semibold text-blue-gray-600">
                    {contact || "N/A"}
</Typography>
                )}
</td>
<td className="py-3 px-5 border-b border-blue-gray-50">
                {isEditing ? (
<Input
                    value={editedUserDetails.vehicle}
onChange={(e) =>handleInputChange(e, "vehicle")}
                    placeholder="Vehicle"
                  />
                ) : (
<Typography className="text-xs font-semibold text-blue-gray-600">
                    {vehicle || "N/A"}
</Typography>
                )}
</td>
<td className="py-3 px-5 border-b border-blue-gray-50">
                {role !== 'admin' ? (
<Select
                    value={role}
onChange={(newRole) =>handleRoleChange(_id, newRole)}
className="w-full"
>
<Option value="user">User</Option>
<Option value="moderator">Moderator</Option>
</Select>
                ) : (
<p>Admin</p>
```

```
                )}
</td>
<td className="py-3 px-5 border-b border-blue-gray-50">
<Chip
                 variant="gradient"
color={verified === "true" ? "green" : "blue-gray"}
                 value={verified === "true" ? "Verified" : "Not Verified"}
className="py-0.5 px-2 text-[11px] font-medium w-fit"
                />
</td>
<td className="py-3 px-5 flex flex-row border-b border-blue-gray-50 justify-around">
                {isEditing ? (
<>
<Button
color="green"
                    size="sm"
onClick={() =>handleSaveUser(_id)}
>
                    Save
</Button>
<Button
color="red"
                    size="sm"
onClick={handleCancelEdit}
>
                    Cancel
</Button>
</>
                ) : (
<>
<Typography
                    as="a"
href="#"
```

```
className="text-xs font-semibold text-blue-gray-600 cursor-pointer"

onClick={() =>handleViewUser({ _id, profilePicture, firstName, lastName, email, role, verified, contact, vehicle
})}

>

                    View

</Typography>

                  {canEdit&& (

<Typography

                     as="a"

href="#"

className="text-xs font-semibold text-blue-600 cursor-pointer"

onClick={() =>handleEditUser({ _id, profilePicture, firstName, lastName, email, role, verified, contact, vehicle })}

>

                     Edit

</Typography>

                  )}

<Typography

                    as="a"

href="#"

className="text-xs font-semibold text-red-600"

>

                    Block

</Typography>

</>

                )}

</td>

</tr>

            );

          })}

</tbody>

</table>

</CardBody>

</Card>
```

```
<div className="flex justify-center gap-2">
<Button
onClick={() =>handlePageChange(currentPage - 1)}
        disabled={currentPage === 1}
>
        Previous
</Button>
<Typography className="flex items-center gap-2">
        Page {currentPage} of {totalPages}
</Typography>
<Button
onClick={() =>handlePageChange(currentPage + 1)}
        disabled={currentPage === totalPages}
>
        Next
</Button>
</div>

<UserModal
      user={selectedUser}
      open={isModalOpen}
handleClose={handleCloseModal}
    />
</div>
 );
}

export default UserSection;
```