VSCODE_PRINT_SCRIPT_TAGS

Folder backend

13 printable files

(file list disabled)

backend\.env

```
PORT=8000

MONGO_URL=mongodb+srv://uccmaniruddinkhan:uccmaniruddinkhan@arbnb.if1nwqj.mongodb.net/?retryWrites=true&w=majority

JWT_SECRET=hakolkfmfowwerdlcs



TWILIO_SID=AYajjmpDLunawN9mRtBUbWAMSNG9on1NRL
TWILIO_TOKEN=589b8e79a5b00d12bb44cde7a21e17ba
TWILIO_MOBILE_NUM=+18787887222

# pranshoo.rathore.freelance@testbook.com
```

backend\.gitignore

```
# Logs
logs
*.log
npm-debug.log*
yarn-debug.log*
yarn-error.log*
pnpm-debug.log*
lerna-debug.log*

.env
node_modules
dist
dist-ssr
*.local

# Editor directories and files
.vscode/*
!.vscode/extensions.json
.idea
.DS_Store
*.suo
*.ntvs*
*.njsproj
*.sln
*.sw?
```

backend\controllers\authController.js

```
1
2  import userModel from "../model/userModel.js";
3  import { hassPassword, comparePassword } from '../helpers/authHelper.js'
4  import JWT from 'jsonwebtoken';
5  import twilio from 'twilio';
6
7
8
9
10 const registerController = async (req, res) => {
11     try {
12         const { firstname, lastname, email, mobno, password } = req.body
13         // for validations use
14         if (!firstname) {
15             return res.send({ message: 'Firstname is required' })
16         }
17         if (!lastname) {
18             return res.send({ message: 'Lastname is required' })
19         }
20         if (!email) {
21             return res.send({ message: 'Email is required' })
22         }
23         if (!mobno) {
24             return res.send({ message: 'Phone is required' })
25         }
26         if (!password) {
27             return res.send({ message: 'Password is required' })
28         }
29         // check user
30         const existingUser = await userModel.findOne({ email })
31         // existing user
32         if (existingUser) {
33             return res.status(200).send({
34                 status: 'false',
35                 message: 'Already register please login'
36             })
37         }
38         console.log("register checking", req.body)
39         // resgister user
40         const hashedPassword = await hassPassword(password)
41         //save
42         const user = await new userModel({ firstname, lastname, email, mobno, password:
   hashedPassword, }).save()
43         res.status(201).send({
44             success: true,
45             message: 'User registered succesfully',
46             user,
47         })
48     } catch (error) {
49         console.log(error)
50         res.status(500).send({
51             message: "Internal server error",
52             error
53         })
54     }
55 };
56
57 export default registerController;
58
```

```
59
60  export const loginController = async (req, res) => {
61      try {
62          const { email, password } = req.body;
63          // validation
64          if (!email || !password) {
65              return res.status(404).send({
66                  success: false,
67                  message: "Invalid email and password"
68              })
69          }
70          // checking user
71          const user = await userModel.findOne({ email });
72          if (!user) {
73              throw new Error("Email not found")
74          }
75          const match = await comparePassword(password, user.password)
76          if (!match) {
77              res.status(200).send({
78                  success: false,
79                  message: "Incorrect Password!"
80              })
81          }
82          // token creating
83          const token = await JWT.sign({ _id: user._id }, process.env.JWT_SECRET, {
    expiresIn: '15d' });
84          res.status(200).send({
85              success: true,
86              message: "Loged in Successfully",
87              user: {
88                  name: user.firstname + " " + user.lastname,
89                  email: user.email,
90                  phone: user.mobno,
91              },
92              token,
93          });
94      } catch (error) {
95          console.log(error)
96          res.status(500).send({
97              success: false,
98              message: "Login in error",
99              error
100         })
101     }
102 }
103
104 export const forgetPasswordController = async (req, res) => {
105     try {
106         const { email, mobno, newPassword } = req.body;
107         if (!email) {
108             return res.status(401).json({ "message": "Email is required" })
109         }
110         if (!mobno) {
111             return res.status(401).json({ "message": "answer is required" })
112         }
113         if (!newPassword) {
114             return res.status(401).json({ "message": "New Password is required" })
115         }
116
117         // check user
```

```
        const user = await userModel.findOne({ email, mobno })

        // validation
        if (!user) {
            return res.status(404).send({
                success: false,
                message: "User not found!"
            })
        }

        const hashed = await hassPassword(newPassword)
        await userModel.findByIdAndUpdate(user._id, { password: hashed })
        res.status(200).send({
            success: true,
            message: "Password reset successfully",
        })

    } catch (error) {
        console.log(error)
        res.status(500).send({
            success: false,
            message: "Forget password failed!",
            error
        })
    }
}

// test controller
export const testController = (req, res) => {
    res.send("test controller working fine!!!")
}




// using otp login controller


const AC_twilio = "AC08b8b1b2a69da9c53c0bfa4e5d2fd22f";
const twilio_token = "589b8e79a5b00d12bb44cde7a21e17ba";
const twilio_mobno = "+18787887222";

const twilioClient = twilio(AC_twilio, twilio_token);

export const otpLoginController = async (req, res) => {
    try {
        const { mobno } = req.body;
        const user = await userModel.findOne({ mobno });
        if (!user) {
            return res.status(404).send({
                success: false,
                message: "Mobile number doesnot exist"
            });
        }
        const otp = Math.floor(1000 + Math.random() * 9000);
        await twilioClient.messages.create({
            body: `Your OTP is ${otp}`,
            from: twilio_mobno,
            to: user.mobno,
```

```
        });
        res.status(200).send({
            success: true,
            message: 'OTP sent successfully',
        })
    } catch (error) {
        console.log(error)
        res.status(404).send({
            success: false,
            message: 'OTP Login Failed!',
            error
        })
    }
}
```

backend\controllers\placeController.js

```
import placeModel from "../model/placeModel.js";


export const createPlaceController = async (req, res) => {
    try {
        const { title, address, photos, description, price, perks, extrainfo, checkin, checkout, maxguest } = req.body;

        const newPlace = new placeModel({
            owner: req.user._id, title, address, photos, description, perks, price, extrainfo, checkin, checkout, maxguest
        });
        await newPlace.save();
        res.status(201).json({ message: 'Place listing successfully', data: newPlace });
    } catch (error) {
        console.log(error)
        res.status(400).send({
            success: false,
            message: "Something went wrong",
            error

        })
    }
}



// get All place added by particullar user
export const getPlaceController = async (req, res) => {
    try {
        const userId = req.user._id;
        const places = await placeModel.find({ owner: userId }).sort({ createdAt: -1 });
        res.status(200).json({ places });
    } catch (error) {
        console.error('Error fetching user places:', error);
        return res.status(500).json({ message: 'Internal server error' });
    }
}

// get single place using id added by particullar user
export const getSinPlaceController = async (req, res) => {
```

```
    try {
        const { id } = req.params;
        const place = await placeModel.findById(id);
        res.status(200).json({ place });
    } catch (error) {
        console.error('Error fetching user places:', error);
        return res.status(500).json({ message: 'Internal server error' });
    }
}



// get all place anyone can see

export const getAnyPlacecontroller = async (req, res) => {
    try {
        const allPlaces = await placeModel.find().sort({ createdAt: -1 });
        res.status(200).json({
            success: true,
            message: "All places fetched successfully",
            places: allPlaces,
        });
    } catch (error) {
        console.log(error)
        res.status(401).send({
            success: false,
            message: "something went wrong",
            error
        })
    }
}


// update controller
export const updatePlaceController = async (req, res) => {
    try {
        const { id } = req.params;
        const {
            title,
            address,
            photos,
            description,
            perks,
            price,
            extrainfo,
            checkin,
            checkout,
            maxguest
        } = req.body;

        const updatedPlace = await placeModel.findByIdAndUpdate(
            id,
            {
                title,
                address,
                photos,
                description,
                perks,
                extrainfo,
                price,
```

```
                checkin,
                checkout,
                maxguest
            },
            { new: true }
        );

        if (!updatedPlace) {
            return res.status(404).json({
                success: false,
                message: "No Place found with the given ID"
            });
        }

        return res.json({
            success: true,
            message: "Place updated successfully",
            place: updatedPlace
        });
    } catch (error) {
        res.status(500).json({
            success: false,
            message: "An error occurred while updating the place",
            error
        });
    }
}


// get single place controller
export const getSinglePlaceController = async (req, res) => {
    try {
        const { id } = req.params;
        res.json(await placeModel.findById(id))
    } catch (error) {
        res.status(400).send({
            success: false,
            message: `No Place found with the given ID`,
            error
        })
    }
}



// for delete place

export const deletePlaceController = async (req, res) => {
    try {
        const { id } = req.params;
        const deletedPlace = await placeModel.findByIdAndDelete(id);

        if (!deletedPlace) {
            return res.status(404).json({
                success: false,
                message: "No Place found with the given ID",
            });
        }

        return res.json({
```

```
            success: true,
            message: "Place deleted successfully",
        });
    } catch (error) {
        res.status(500).json({
            success: false,
            message: "An error occurred while deleting the place",
            error,
        });
    }
};
```

backend\db.js

```
import mongoose from 'mongoose';
import colors from 'colors'

const connectDB = async () => {
    try {
        const conn = await mongoose.connect(process.env.MONGO_URL);
        console.log(`Database connected Succesfully
${conn.connection.host}`.bgGreen.white);
    } catch (error) {
        console.log(`Error in connection ${error}`.bgRed.white)
    }
}


export default connectDB;
```

backend\helpers\authHelper.js

```
import bcrypt from 'bcrypt';


export const hassPassword = async (password) => {
    try {
        const saltRounds = 10;
        const hassedPassword = await bcrypt.hash(password, saltRounds);
        return hassedPassword;
    } catch (error) {
        console.log(error)
    }
}

// function for comparing

export const comparePassword = async (password, hassedPassword) => {
    return bcrypt.compare(password, hassedPassword);
};
```

backend\midlewares\authMidlewares.js

```
import JWT from 'jsonwebtoken';
import userModel from '../model/userModel.js';

// protected routes using JWT

export const requireSignIn = async (req, res, next) => {
    try {
        const decode = JWT.verify(req.headers.authorization, process.env.JWT_SECRET);
        req.user = decode;
        next();
    } catch (error) {
        console.log(error)
    }
}


export const isAdmin = async (req, res, next) => {
    try {
        const user = await userModel.findById(req.user._id);
        if (user.role !== 1) {
            return res.status(401).send({
                success: false,
                message: 'You are not authorized to perform this action'
            })
        } else {
            next();
        }
    } catch (error) {
        console.log(error)
    }
}
```

backend\model\placeModel.js

```
import { Timestamp } from 'mongodb';
import mongoose from 'mongoose';

const placeSchema = new mongoose.Schema({
    owner: { type: mongoose.Schema.Types.ObjectId, ref: 'user' },
    title: { type: String, required: true },
    address: { type: String, required: true },
    photos: [String],
    description: String,
    extrainfo: String,
    price: Number,
    checkin: String,
    checkout: String,
    maxguest: Number,
    perks: [String],
}, { timestamps: true });

export default mongoose.model('place', placeSchema);
```

backend\model\userModel.js

```
import mongoose from 'mongoose';


const userSchema = new mongoose.Schema({
    firstname: {
        type: String,
        required: true
    },
    lastname: {
        type: String,
        require: true
    },
    email: {
        type: String,
        unique: [true, 'Email already exist'],
        required: true
    },
    mobno: {
        type: Number,
        required: [true],
        unique: true,
    },
    password: {
        type: String,
        required: true,
    },
}, { timestamps: true })

export default mongoose.model('users', userSchema)
```

backend\package.json

```
{
  "name": "backend",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "type": "module",
  "scripts": {
    "start": "nodemon server.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "bcrypt": "^5.1.0",
    "colors": "^1.4.0",
    "cors": "^2.8.5",
    "dotenv": "^16.3.1",
    "express": "^4.18.2",
    "jsonwebtoken": "^9.0.1",
    "mongodb": "^5.7.0",
    "mongoose": "^7.4.2",
```

```
    "nodemon": "^3.0.1",
    "twilio": "^4.15.0"
  }
}

```

backend\routes\authRoutes.js

```
import express from 'express';
import registerController, { forgetPasswordController, loginController, otpLoginController, testController } from '../controllers/authController.js';
import { isAdmin, requireSignIn } from '../midlewares/authMidlewares.js';

// rest object
const router = express.Router();

// routing
// register || POST Method
router.post('/register', registerController);

// Login Routes
router.post('/login', loginController);

// using otp
router.post('/otp-login', otpLoginController);

// Forgot password Rutes
router.post("/forget-password", forgetPasswordController)

// test routes
router.get('/test', requireSignIn, isAdmin, testController);

// protected routes
router.get('/user-auth', requireSignIn, (req, res) => {
    res.status(200).send({ ok: true });
})

router.get('/admin-auth', requireSignIn, isAdmin, (req, res) => {
    res.status(200).send({ ok: true });
});
export default router;
```

backend\routes\placeRoute.js

```
import express from "express";
import { requireSignIn } from "../midlewares/authMidlewares.js";
import { createPlaceController, deletePlaceController, getAnyPlacecontroller, getPlaceController, getSinPlaceController, getSinglePlaceController, updatePlaceController } from "../controllers/placeController.js";

// rest object
const router = express.Router();

// create place
router.post('/create-place', requireSignIn, createPlaceController)

```

```
// update place
router.put('/update-place/:id', requireSignIn, updatePlaceController)

// get-place
router.get('/get-place', requireSignIn, getPlaceController)

// get All place
router.get('/get-place-all', getAnyPlacecontroller)


// get single place without authentication
router.get('/single-place/:id', getSinPlaceController)

// get single place using id with authentication
router.get('/get-place/:id', requireSignIn, getSinglePlaceController)

// for delete
router.delete('/delete-place:id', requireSignIn, deletePlaceController)


export default router;

```

backend\server.js

```
import express from 'express';
import colors from 'colors'
import dotenv from 'dotenv'
import cors from 'cors'
import connectDB from './db.js';
import authRoutes from "./routes/authRoutes.js";
import placeRoutes from './routes/placeRoute.js'

// config dotenv
dotenv.config()

// datbase connection
connectDB();


// Rest object
const app = express();
const PORT = process.env.PORT || 8000;

// midlewares
app.use(express.json());
app.use(cors({
    credentials: true,
    origin: 'http://localhost:5173',
}));

// Routes
app.use('/api/arrbnb/v1/auth', authRoutes);
app.use('/api/arrbnb/v1/place', placeRoutes);

// rest api
app.get('/', (req, res) => {
    res.send({
```

```
        message: "Welcome to the Arbnb-clone API"
    })
})



app.listen(PORT, () => {
    console.log(`Server is running on port ${PORT}`.white.bgCyan)
})

```