

VSCODE_PRINT_SCRIPT_TAGS

Folder backend

13 printable files

(file list disabled)

backend\.env

```
1  PORT=8000
2
3  MONGO_URL=mongodb+srv://uccmaniruddinkhan:uccmaniruddinkhan@arbnb.if1nwqj.mongodb.net/?
  retryWrites=true&w=majority
4
5  JWT_SECRET=hakolkfmfowwerdlcs
6
7
8  TWILIO_SID=AYajjmplDunawN9mRtBUBWAMSNG9on1NRL
9  TWILIO_TOKEN=589b8e79a5b00d12bb44cde7a21e17ba
10 TWILIO_MOBILE_NUM=+18787887222
11
12 # pranshoo.rathore.freelance@testbook.com
```

backend\.gitignore

```
1  # Logs
2  logs
3  *.log
4  npm-debug.log*
5  yarn-debug.log*
6  yarn-error.log*
7  pnpm-debug.log*
8  lerna-debug.log*
9
10 .env
11 node_modules
12 dist
13 dist-ssr
14 *.local
15
16 # Editor directories and files
17 .vscode/*
18 !.vscode/extensions.json
19 .idea
20 .DS_Store
21 *.suo
22 *.ntvs*
23 *.njsproj
24 *.sln
25 *.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     // register 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, {
84 expiresIn: '15d' });
85         res.status(200).send({
86             success: true,
87             message: "Logged in Successfully",
88             user: {
89                 name: user.firstname + " " + user.lastname,
90                 email: user.email,
91                 phone: user.mobno,
92             },
93             token,
94         });
95     } catch (error) {
96         console.log(error)
97         res.status(500).send({
98             success: false,
99             message: "Login in error",
100             error
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         // check user
117

```

```

118     const user = await userModel.findOne({ email, mobno })
119
120     // validation
121     if (!user) {
122         return res.status(404).send({
123             success: false,
124             message: "User not found!"
125         })
126     }
127
128     const hashed = await hassPassword(newPassword)
129     await userModel.findByIdAndUpdate(user._id, { password: hashed })
130     res.status(200).send({
131         success: true,
132         message: "Password reset successfully",
133     })
134
135 } catch (error) {
136     console.log(error)
137     res.status(500).send({
138         success: false,
139         message: "Forget password failed!",
140         error
141     })
142 }
143 }
144
145 // test controller
146 export const testController = (req, res) => {
147     res.send("test controller working fine!!!")
148 }
149
150
151
152
153
154 // using otp login controller
155
156
157 const AC_twilio = "AC08b8b1b2a69da9c53c0bfa4e5d2fd22f";
158 const twilio_token = "589b8e79a5b00d12bb44cde7a21e17ba";
159 const twilio_mobno = "+18787887222";
160
161 const twilioClient = twilio(AC_twilio, twilio_token);
162
163 export const otpLoginController = async (req, res) => {
164     try {
165         const { mobno } = req.body;
166         const user = await userModel.findOne({ mobno });
167         if (!user) {
168             return res.status(404).send({
169                 success: false,
170                 message: "Mobile number doesnot exist"
171             });
172         }
173         const otp = Math.floor(1000 + Math.random() * 9000);
174         await twilioClient.messages.create({
175             body: `Your OTP is ${otp}`,
176             from: twilio_mobno,
177             to: user.mobno,

```

```

178     });
179     res.status(200).send({
180         success: true,
181         message: 'OTP sent successfully',
182     })
183 } catch (error) {
184     console.log(error)
185     res.status(404).send({
186         success: false,
187         message: 'OTP Login Failed!',
188         error
189     })
190 }
191 }

```

backend\controllers\placeController.js

```

1  import placeModel from "../model/placeModel.js";
2
3
4  export const createPlaceController = async (req, res) => {
5      try {
6          const { title, address, photos, description, price, perks, extrainfo, checkin,
9          checkout, maxguest } = req.body;
7
8          const newPlace = new placeModel({
9              owner: req.user._id, title, address, photos, description, perks, price,
10             extrainfo, checkin, checkout, maxguest
11         });
12         await newPlace.save();
13         res.status(201).json({ message: 'Place listing successfully', data: newPlace });
14     } catch (error) {
15         console.log(error)
16         res.status(400).send({
17             success: false,
18             message: "Something went wrong",
19             error
20         })
21     }
22 }
23
24
25
26 // get All place added by particular user
27 export const getPlaceController = async (req, res) => {
28     try {
29         const userId = req.user._id;
30         const places = await placeModel.find({ owner: userId }).sort({ createdAt: -1 });
31         res.status(200).json({ places });
32     } catch (error) {
33         console.error('Error fetching user places:', error);
34         return res.status(500).json({ message: 'Internal server error' });
35     }
36 }
37
38 // get single place using id added by particular user
39 export const getSinPlaceController = async (req, res) => {

```

```

40   try {
41     const { id } = req.params;
42     const place = await placeModel.findById(id);
43     res.status(200).json({ place });
44   } catch (error) {
45     console.error('Error fetching user places:', error);
46     return res.status(500).json({ message: 'Internal server error' });
47   }
48 }
49
50
51
52 // get all place anyone can see
53
54 export const getAnyPlacecontroller = async (req, res) => {
55   try {
56     const allPlaces = await placeModel.find().sort({ createdAt: -1 });
57     res.status(200).json({
58       success: true,
59       message: "All places fetched successfully",
60       places: allPlaces,
61     });
62   } catch (error) {
63     console.log(error)
64     res.status(401).send({
65       success: false,
66       message: "something went wrong",
67       error
68     })
69   }
70 }
71
72
73 // update controller
74 export const updatePlaceController = async (req, res) => {
75   try {
76     const { id } = req.params;
77     const {
78       title,
79       address,
80       photos,
81       description,
82       perks,
83       price,
84       extrainfo,
85       checkin,
86       checkout,
87       maxguest
88     } = req.body;
89
90     const updatedPlace = await placeModel.findByIdAndUpdate(
91       id,
92       {
93         title,
94         address,
95         photos,
96         description,
97         perks,
98         extrainfo,
99         price,

```

```

100         checkin,
101         checkout,
102         maxguest
103     },
104     { new: true }
105 );
106
107 if (!updatedPlace) {
108     return res.status(404).json({
109         success: false,
110         message: "No Place found with the given ID"
111     });
112 }
113
114 return res.json({
115     success: true,
116     message: "Place updated successfully",
117     place: updatedPlace
118 });
119 } catch (error) {
120     res.status(500).json({
121         success: false,
122         message: "An error occurred while updating the place",
123         error
124     });
125 }
126 }
127
128
129 // get single place controller
130 export const getSinglePlaceController = async (req, res) => {
131     try {
132         const { id } = req.params;
133         res.json(await placeModel.findById(id))
134     } catch (error) {
135         res.status(400).send({
136             success: false,
137             message: `No Place found with the given ID`,
138             error
139         })
140     }
141 }
142
143
144
145 // for delete place
146
147 export const deletePlaceController = async (req, res) => {
148     try {
149         const { id } = req.params;
150         const deletedPlace = await placeModel.findByIdAndDelete(id);
151
152         if (!deletedPlace) {
153             return res.status(404).json({
154                 success: false,
155                 message: "No Place found with the given ID",
156             });
157         }
158
159         return res.json({

```

```

160         success: true,
161         message: "Place deleted successfully",
162     });
163 } catch (error) {
164     res.status(500).json({
165         success: false,
166         message: "An error occurred while deleting the place",
167         error,
168     });
169 }
170 };

```

backend\db.js

```

1 import mongoose from 'mongoose';
2 import colors from 'colors'
3
4 const connectDB = async () => {
5     try {
6         const conn = await mongoose.connect(process.env.MONGO_URL);
7         console.log(`Database connected Successfully
8         ${conn.connection.host}`.bgGreen.white);
9     } catch (error) {
10         console.log(`Error in connection ${error}`.bgRed.white)
11     }
12 }
13
14 export default connectDB;

```

backend\helpers\authHelper.js

```

1 import bcrypt from 'bcrypt';
2
3
4 export const hassPassword = async (password) => {
5     try {
6         const saltRounds = 10;
7         const hassedPassword = await bcrypt.hash(password, saltRounds);
8         return hassedPassword;
9     } catch (error) {
10         console.log(error)
11     }
12 }
13
14 // function for comparing
15
16 export const comparePassword = async (password, hassedPassword) => {
17     return bcrypt.compare(password, hassedPassword);
18 };

```

backend\middlewares\authMiddleware.js


```

1 import JWT from 'jsonwebtoken';
2 import userModel from '../model/userModel.js';
3
4 // protected routes using JWT
5
6 export const requireSignIn = async (req, res, next) => {
7   try {
8     const decode = JWT.verify(req.headers.authorization, process.env.JWT_SECRET);
9     req.user = decode;
10    next();
11  } catch (error) {
12    console.log(error)
13  }
14 }
15
16
17 export const isAdmin = async (req, res, next) => {
18   try {
19     const user = await userModel.findById(req.user._id);
20     if (user.role !== 1) {
21       return res.status(401).send({
22         success: false,
23         message: 'You are not authorized to perform this action'
24       });
25     } else {
26       next();
27     }
28   } catch (error) {
29     console.log(error)
30   }
31 }

```

backend\model\placeModel.js

```

1 import { Timestamp } from 'mongoose';
2 import mongoose from 'mongoose';
3
4 const placeSchema = new mongoose.Schema({
5   owner: { type: mongoose.Schema.Types.ObjectId, ref: 'user' },
6   title: { type: String, required: true },
7   address: { type: String, required: true },
8   photos: [String],
9   description: String,
10  extrainfo: String,
11  price: Number,
12  checkin: String,
13  checkout: String,
14  maxguest: Number,
15  perks: [String],
16 }, { timestamps: true });
17
18 export default mongoose.model('place', placeSchema);
19
20
21

```

backend\model\userModel.js

```
1 import mongoose from 'mongoose';
2
3
4 const userSchema = new mongoose.Schema({
5   firstname: {
6     type: String,
7     required: true
8   },
9   lastname: {
10    type: String,
11    require: true
12  },
13  email: {
14    type: String,
15    unique: [true, 'Email already exist'],
16    required: true
17  },
18  mobno: {
19    type: Number,
20    required: [true],
21    unique: true,
22  },
23  password: {
24    type: String,
25    required: true,
26  },
27 }, { timestamps: true })
28
29 export default mongoose.model('users', userSchema)
```

backend\package.json

```
1 {
2   "name": "backend",
3   "version": "1.0.0",
4   "description": "",
5   "main": "index.js",
6   "type": "module",
7   "scripts": {
8     "start": "nodemon server.js",
9     "test": "echo \"Error: no test specified\" && exit 1"
10  },
11  "keywords": [],
12  "author": "",
13  "license": "ISC",
14  "dependencies": {
15    "bcrypt": "^5.1.0",
16    "colors": "^1.4.0",
17    "cors": "^2.8.5",
18    "dotenv": "^16.3.1",
19    "express": "^4.18.2",
20    "jsonwebtoken": "^9.0.1",
21    "mongodb": "^5.7.0",
22    "mongoose": "^7.4.2",
```

```

23 |     "nodemon": "^3.0.1",
24 |     "twilio": "^4.15.0"
25 | }
26 | }
27 |

```

backend\routes\authRoutes.js

```

1 | import express from 'express';
2 | import registerController, { forgetPasswordController, loginController, otpLoginController,
  testController } from '../controllers/authController.js';
3 | import { isAdmin, requireSignIn } from '../middlewares/authMiddleware.js';
4 |
5 | // rest object
6 | const router = express.Router();
7 |
8 | // routing
9 | // register || POST Method
10 | router.post('/register', registerController);
11 |
12 | // Login Routes
13 | router.post('/login', loginController);
14 |
15 | // using otp
16 | router.post('/otp-login', otpLoginController);
17 |
18 | // Forgot password Rutes
19 | router.post("/forget-password", forgetPasswordController)
20 |
21 | // test routes
22 | router.get('/test', requireSignIn, isAdmin, testController);
23 |
24 | // protected routes
25 | router.get('/user-auth', requireSignIn, (req, res) => {
26 |     res.status(200).send({ ok: true });
27 | })
28 |
29 | router.get('/admin-auth', requireSignIn, isAdmin, (req, res) => {
30 |     res.status(200).send({ ok: true });
31 | });
32 | export default router;

```

backend\routes\placeRoute.js

```

1 | import express from "express";
2 | import { requireSignIn } from "../middlewares/authMiddleware.js";
3 | import { createPlaceController, deletePlaceController, getAnyPlacecontroller,
  getPlaceController, getSinPlaceController, getSinglePlaceController, updatePlaceController
  } from "../controllers/placeController.js";
4 |
5 | // rest object
6 | const router = express.Router();
7 |
8 | // create place
9 | router.post('/create-place', requireSignIn, createPlaceController)
10 |

```

```

11 // update place
12 router.put('/update-place/:id', requireSignIn, updatePlaceController)
13
14 // get-place
15 router.get('/get-place', requireSignIn, getPlaceController)
16
17 // get All place
18 router.get('/get-place-all', getAnyPlacecontroller)
19
20
21 // get single place without authentication
22 router.get('/single-place/:id', getSinPlaceController)
23
24 // get single place using id with authentication
25 router.get('/get-place/:id', requireSignIn, getSinglePlaceController)
26
27 // for delete
28 router.delete('/delete-place:id', requireSignIn, deletePlaceController)
29
30
31 export default router;
32

```

backend\server.js

```

1 import express from 'express';
2 import colors from 'colors'
3 import dotenv from 'dotenv'
4 import cors from 'cors'
5 import connectDB from './db.js';
6 import authRoutes from './routes/authRoutes.js';
7 import placeRoutes from './routes/placeRoute.js'
8
9 // config dotenv
10 dotenv.config()
11
12 // database connection
13 connectDB();
14
15
16 // Rest object
17 const app = express();
18 const PORT = process.env.PORT || 8000;
19
20 // midlewares
21 app.use(express.json());
22 app.use(cors({
23   credentials: true,
24   origin: 'http://localhost:5173',
25 }));
26
27 // Routes
28 app.use('/api/arrbnb/v1/auth', authRoutes);
29 app.use('/api/arrbnb/v1/place', placeRoutes);
30
31 // rest api
32 app.get('/', (req, res) => {
33   res.send({

```

```
34         message: "Welcome to the Arbnb-clone API"
35     })
36 })
37
38
39
40 app.listen(PORT, () => {
41     console.log(`Server is running on port ${PORT}`.white.bgCyan)
42 })
43
44
```